

iceMASTER™ Software Version 3.3 Supplement

```

Module:TA Source File:takf511.aom
90 /* scan through nodes in list */
91 p_list = list_head;
92 while (p_list != NULL) {
93     lj = p_list->list_uival;
94     p_list = p_list->list_p_nxt;
95 } /*while p_list*/
96
97 iteration += 1.1;
98 main_loop_factor++;
99
100 End_Loop: ; /* (for demo's -- in case lines inserted/deleted) */
101 } /*while TRUE*/
102
103 } /*main*/
104 } /*main*/
Enter:Change MATCH Ctrl-B:Browse
X:008E uint ai 10
X:0090 uint aj 21
X:008E uchar ary[ail 0x00 ' ' 0
X:0084 [10]uchar ary [10]uchar
X:0084 uchar ..... [ 0] 0x03 ' ' 3
X:00A5 ->struct List list_head [ 1] 0x05 ' ' 5
X:009E ->struct List ..... 0x07 ' ' 7
X:0099 ->struct List ..... 9
X:009C Addr Addr Data Type Value ..... 11
X:009C X:0097 ..... 13
X:0097 X:0097 X:0092 struct List struct List ..... 15
X:0097 X:0097 X:0092 uint list_uival 3 ..... 17
Enter X:0097 X:0094 ->struct List list_p_nxt (null) ..... 19
X:008C Enter:Change Ctrl-B:Browse ..... 21
X:008D uchar p_list (null)
X:00A8 ->struct List *p_list struct List
C:0000?struct List ..... list_uival 517
C:0000?uint ..... list_p_nxt ? :FFFF(CEFFFF)
C:0002?>struct List ..... [3]struct List
X:0092 [3]struct List nodes [0] struct List
X:0092 struct List ..... list_uival 3
X:0094 ->struct List ..... list_p_nxt (null)
X:0097 struct List [1] struct List
X:0097 uint ..... list_uival 7
X:0099 ->struct List ..... list_p_nxt X:0092
X:009C struct List [2] struct List
X:009C uint ..... list_uival 11
X:009E ->struct List ..... list_p_nxt X:0097
X:00A5 ->struct List list head X:009C
X:0080 float iteration 2.2

```

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit

© 1993 by MetaLink Corporation

All rights are reserved.

This manual may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without the prior agreement and written permission of MetaLink Corporation.

ICE-HOUSE[®], MetaICE-CHEST[®], MetaICE[®], MicroICE[®], and MicroICE[®] +, and the MetaLink logo are registered trademarks of MetaLink Corporation.

iceMASTER-PE[™], iceMASTER[™], ICELink[™], MetaLink[™], and the combination of MetaICE[®], iceMASTER-PE[™], iceMASTER[™], MicroICE[®], or MicroICE[®] + and an alphabetic or numeric suffix are claimed trademarks of MetaLink Corporation and may only be used to describe MetaLink products.

National Semiconductor[®] and National Semiconductor Corp.[®] are registered trademarks of National Semiconductor Corporation.

MOLE[™] is a trademark of National Semiconductor Corporation.

IBM[®] is a registered trademark of IBM Corporation.

PC-DOS is a trademark of IBM Corporation.

MS-DOS[®] is a registered trademark of Microsoft Corporation.

Windows[™] is a trademark of Microsoft Corporation.

MCS[®] and Intel[®] are registered trademarks of Intel Corporation.

Archimedes[™] is a trademark of Archimedes Software, Inc.

Franklin[™] is a trademark of Franklin Software, Inc.

MetaLink Corporation reserves the right to make improvements in the products described in this manual as well as the manual itself at any time and without notice.

Table of Contents

Chapter 1: Introduction

1-1

Chapter 2: Tutorial

2-1

Program TA.C	2-2
Watch Window	2-4
Browsing	2-9
Display/Alter Expression Command	2-12
Source/Symbols Commands	2-16
Improved Support for "Unsupported" Programs	2-19

Chapter 3: Reference

3-1

Expression Input	3-2
Identifiers	3-2
Constants	3-2
Integer Constants	3-2
Explicit Radix	3-2
No Explicit Radix	3-4
Floating-Point Constants	3-5
Character Constants	3-5
String Constants	3-6
Expressions	3-6
Primary	3-6
Postfix	3-8
Unary	3-8
Cast	3-8
Multiplicative	3-9
Additive	3-9
Display Formats	3-10
Addresses	3-10
Type Names	3-11
Fundamental Types	3-11

CC-1	Derived/Aggregate Types	3-12
PC-1	Values	3-12
CC-1	Fundamental Types	3-12
PC-1	Pointers	3-13
PC-1	Generic Pointers	3-13
PC-1	Memory-Specific Pointers	3-13
PC-1	NULL Pointers	3-13
PC-1	Structures, Unions and Arrays	3-14
PC-1	Functions	3-14
PC-1	Configure Options Expressions Command	3-15
PC-1	Configure Options Expressions Radix	3-15
PC-1	Configure Options Expressions Unknown data type size	3-16
PC-1	Configure Options Expressions Browse Windows Stack	3-16
PC-1	Configure Options Expressions Struct/Union Indentation	3-17
PC-1	Configure Options Expressions Member Names	3-18
PC-1	Configure Options Expressions Array Subscript Values	3-19
PC-1	Configure Options Expressions unknown/untyped	
PC-1	Configure Options Expressions short/ushort	
PC-1	Configure Options Expressions int/uint	
PC-1	Configure Options Expressions long/ulong	
PC-1	Configure Options Expressions float	
PC-1	Configure Options Expressions double	3-20
PC-1	Configure Options Expressions Command Combinations	3-21
PC-1	Watch Window	3-22
PC-1	Commands	3-22
PC-1	Display Format	3-22
PC-1	Browsing	3-23
PC-1	Source/Symbols Displays	3-24
PC-1	Display/Alter Expression	3-25
PC-1	Change Box	3-26
PC-1	Browse Windows	3-27

Compilers	3-29
Keil/Franklin C51	3-29
Recommended Compilation Options	3-29
Recommended Linking Options	3-29
Pointers	3-29
NULL	3-29
'typedef' Names	3-30
Enumeration Types	3-30
'double' Types	3-30
Batch "Make" Files	3-30

IAR/Archimedes C-51	3-31
Recommended Compilation Options	3-31
Recommended Linking Options	3-31
Pointers	3-31
NULL	3-31
Reading	3-31
Writing	3-31
'typedef' Names	3-32
Enumeration Types	3-32
'double' Types	3-32
Batch "Make" Files	3-32

Appendix A: Source Window & Assembler/Disassembler **A-1**

Source Window	A-1
Source Window Control	A-1
Browse	A-1
Change Module	A-2
Mode	A-2
Mode Code	A-2
Mode Mixed	A-2
Mode HLL	A-2
Label-Synch	A-2
Set Break	A-2

<i>Set Break Toggle</i>	A-2
<i>Set Break Run Until</i>	A-3
<i>Assemble</i>	A-3
<i>Display/Alter Asm/Dasm</i>	A-3
<i>Display/Alter Asm/Dasm Browse</i>	A-3
<i>Display/Alter Asm/Dasm Assemble</i>	A-5
<i>Display/Alter Asm/Dasm Mode</i>	A-5
<i>Display/Alter Asm/Dasm Label-synch</i>	A-5
<i>Display/Alter Asm/Dasm Write</i>	A-6

Appendix B: Virtual Memory Support B-1

<i>Command Line Options</i>	B-1
<i>Misc Virtual Memory Command</i>	B-3

Chapter 1: Introduction

iceMASTER Host Software Version 3.3 contains these major enhancements:

- All address, length or count specifications can now be general expressions, rather than just a single symbol or constant.
- Support for long integer, single-precision (**float**) and double-precision (**double**) floating-point variables
 - Display
 - Input
- Full support for structures, unions, arrays and pointers
 - Display/Change
 - * Watch Window
 - * *Display/Alter|Expression* command
 - * *Source/Symbols|Global* command
 - * *Source/Symbols|Local* command
 - * *Source/Symbols|Address* command
 - * *Source/Symbols|Alpha* command
 - Full structure, union, array & pointer support for these compilers:
 - * Keil/Franklin C51
 - * IAR/Archimedes C-51
 - * (others coming soon)
- Data Structure Browser/Inspector
 - Inspect structures, unions and arrays
 - Follow pointers
 - Change any value at any point along the way
 - Backtrack
- True Expressions in Watch Window (not just static addresses)
 - Indirection through NULL or invalid pointers is flagged in the display.
- OS-Escape ("Shell-Out") to DOS during Emulation
 - OS-Escape Hot Key: **Alt-O**
 - Emulation continues while working in DOS.
 - All DOS memory, except about 7K-10K bytes, available during all OS-Escapes
- Supports all *iceMASTER* hardware products
 - *iceMASTER* Model 200/400 Emulators (MCS-51, COP8, 68HC11 & 68HC05)
 - *iceMASTER*-PE Emulators
 - *iceMASTER*-COP8 Debug Modules

The following features were added in earlier versions of the *iceMASTER* host software. Depending on which *iceMASTER* hardware product you purchased, these features may or may not be documented in the primary manual:

- **Source Window & Assembler/Disassembler Enhancements**
(software version 3.2 – see Appendix A)

- "Source-Only" display mode option in Source Window
- Assembler/Disassembler Window fully-scrollable
- Set permanent break-points directly in Source or Assembler/Disassembler Windows
- Set temporary break-points directly in Source Window

- **Virtual Memory Support**
(software version 3.1 – see Appendix B)

- Allows host software to use Expanded Memory for large programs

Chapter 2: Tutorial

Here is short tour through some of the new features in the *iceMASTER* host software. The tutorial is based primarily on the C program **TA.C** shown on the following pages.

```
1  /* Defines and on compilation command line:
2
3  4  *
5  6  *
6  7  * LARGE memory model
7  8  * COMPACT compact memory model
8  9  * SMALL small memory model
9  10 *
10 11 *
11 12 * 80386 architecture
12 13 * All
13 14 *
14 15 *
15 16 *
16 17 *
17 18 *
18 19 *
19 20 *
20 21 *
21 22 *
22 23 *
23 24 *
24 25 *
25 26 *
26 27 *
27 28 *
28 29 *
29 30 *
30 31 *
31 32 *
32 33 *
33 34 *
34 35 *
35 36 *
36 37 *
37 38 *
38 39 *
39 40 *
40 41 *
41 42 *
42 43 *
43 44 *
44 45 *
45 46 *
46 47 *
47 48 *
48 49 *
49 50 *
50 51 *
51 52 *
52 53 *
53 54 *
54 55 *
55 56 *
56 57 *
57 58 *
58 59 *
59 60 *
60 61 *
61 62 *
62 63 *
63 64 *
64 65 *
65 66 *
66 67 *
67 68 *
68 69 *
69 70 *
70 71 *
71 72 *
72 73 *
73 74 *
74 75 *
75 76 *
76 77 *
77 78 *
78 79 *
79 80 *
80 81 *
81 82 *
82 83 *
83 84 *
84 85 *
85 86 *
86 87 *
87 88 *
88 89 *
89 90 *
90 91 *
91 92 *
92 93 *
93 94 *
94 95 *
95 96 *
96 97 *
97 98 *
98 99 *
99 100 *
100 101 *
101 102 *
102 103 *
103 104 *
104 105 *
105 106 *
106 107 *
107 108 *
108 109 *
109 110 *
110 111 *
111 112 *
112 113 *
113 114 *
114 115 *
115 116 *
116 117 *
117 118 *
118 119 *
119 120 *
120 121 *
121 122 *
122 123 *
123 124 *
124 125 *
125 126 *
126 127 *
127 128 *
128 129 *
129 130 *
130 131 *
131 132 *
132 133 *
133 134 *
134 135 *
135 136 *
136 137 *
137 138 *
138 139 *
139 140 *
140 141 *
141 142 *
142 143 *
143 144 *
144 145 *
145 146 *
146 147 *
147 148 *
148 149 *
149 150 *
150 151 *
151 152 *
152 153 *
153 154 *
154 155 *
155 156 *
156 157 *
157 158 *
158 159 *
159 160 *
160 161 *
161 162 *
162 163 *
163 164 *
164 165 *
165 166 *
166 167 *
167 168 *
168 169 *
169 170 *
170 171 *
171 172 *
172 173 *
173 174 *
174 175 *
175 176 *
176 177 *
177 178 *
178 179 *
179 180 *
180 181 *
181 182 *
182 183 *
183 184 *
184 185 *
185 186 *
186 187 *
187 188 *
188 189 *
189 190 *
190 191 *
191 192 *
192 193 *
193 194 *
194 195 *
195 196 *
196 197 *
197 198 *
198 199 *
199 200 *
200 201 *
201 202 *
202 203 *
203 204 *
204 205 *
205 206 *
206 207 *
207 208 *
208 209 *
209 210 *
210 211 *
211 212 *
212 213 *
213 214 *
214 215 *
215 216 *
216 217 *
217 218 *
218 219 *
219 220 *
220 221 *
221 222 *
222 223 *
223 224 *
224 225 *
225 226 *
226 227 *
227 228 *
228 229 *
229 230 *
230 231 *
231 232 *
232 233 *
233 234 *
234 235 *
235 236 *
236 237 *
237 238 *
238 239 *
239 240 *
240 241 *
241 242 *
242 243 *
243 244 *
244 245 *
245 246 *
246 247 *
247 248 *
248 249 *
249 250 *
250 251 *
251 252 *
252 253 *
253 254 *
254 255 *
255 256 *
256 257 *
257 258 *
258 259 *
259 260 *
260 261 *
261 262 *
262 263 *
263 264 *
264 265 *
265 266 *
266 267 *
267 268 *
268 269 *
269 270 *
270 271 *
271 272 *
272 273 *
273 274 *
274 275 *
275 276 *
276 277 *
277 278 *
278 279 *
279 280 *
280 281 *
281 282 *
282 283 *
283 284 *
284 285 *
285 286 *
286 287 *
287 288 *
288 289 *
289 290 *
290 291 *
291 292 *
292 293 *
293 294 *
294 295 *
295 296 *
296 297 *
297 298 *
298 299 *
299 300 *
300 301 *
301 302 *
302 303 *
303 304 *
304 305 *
305 306 *
306 307 *
307 308 *
308 309 *
309 310 *
310 311 *
311 312 *
312 313 *
313 314 *
314 315 *
315 316 *
316 317 *
317 318 *
318 319 *
319 320 *
320 321 *
321 322 *
322 323 *
323 324 *
324 325 *
325 326 *
326 327 *
327 328 *
328 329 *
329 330 *
330 331 *
331 332 *
332 333 *
333 334 *
334 335 *
335 336 *
336 337 *
337 338 *
338 339 *
339 340 *
340 341 *
341 342 *
342 343 *
343 344 *
344 345 *
345 346 *
346 347 *
347 348 *
348 349 *
349 350 *
350 351 *
351 352 *
352 353 *
353 354 *
354 355 *
355 356 *
356 357 *
357 358 *
358 359 *
359 360 *
360 361 *
361 362 *
362 363 *
363 364 *
364 365 *
365 366 *
366 367 *
367 368 *
368 369 *
369 370 *
370 371 *
371 372 *
372 373 *
373 374 *
374 375 *
375 376 *
376 377 *
377 378 *
378 379 *
379 380 *
380 381 *
381 382 *
382 383 *
383 384 *
384 385 *
385 386 *
386 387 *
387 388 *
388 389 *
389 390 *
390 391 *
391 392 *
392 393 *
393 394 *
394 395 *
395 396 *
396 397 *
397 398 *
398 399 *
399 400 *
400 401 *
401 402 *
402 403 *
403 404 *
404 405 *
405 406 *
406 407 *
407 408 *
408 409 *
409 410 *
410 411 *
411 412 *
412 413 *
413 414 *
414 415 *
415 416 *
416 417 *
417 418 *
418 419 *
419 420 *
420 421 *
421 422 *
422 423 *
423 424 *
424 425 *
425 426 *
426 427 *
427 428 *
428 429 *
429 430 *
430 431 *
431 432 *
432 433 *
433 434 *
434 435 *
435 436 *
436 437 *
437 438 *
438 439 *
439 440 *
440 441 *
441 442 *
442 443 *
443 444 *
444 445 *
445 446 *
446 447 *
447 448 *
448 449 *
449 450 *
450 451 *
451 452 *
452 453 *
453 454 *
454 455 *
455 456 *
456 457 *
457 458 *
458 459 *
459 460 *
460 461 *
461 462 *
462 463 *
463 464 *
464 465 *
465 466 *
466 467 *
467 468 *
468 469 *
469 470 *
470 471 *
471 472 *
472 473 *
473 474 *
474 475 *
475 476 *
476 477 *
477 478 *
478 479 *
479 480 *
480 481 *
481 482 *
482 483 *
483 484 *
484 485 *
485 486 *
486 487 *
487 488 *
488 489 *
489 490 *
490 491 *
491 492 *
492 493 *
493 494 *
494 495 *
495 496 *
496 497 *
497 498 *
498 499 *
499 500 *
500 501 *
501 502 *
502 503 *
503 504 *
504 505 *
505 506 *
506 507 *
507 508 *
508 509 *
509 510 *
510 511 *
511 512 *
512 513 *
513 514 *
514 515 *
515 516 *
516 517 *
517 518 *
518 519 *
519 520 *
520 521 *
521 522 *
522 523 *
523 524 *
524 525 *
525 526 *
526 527 *
527 528 *
528 529 *
529 530 *
530 531 *
531 532 *
532 533 *
533 534 *
534 535 *
535 536 *
536 537 *
537 538 *
538 539 *
539 540 *
540 541 *
541 542 *
542 543 *
543 544 *
544 545 *
545 546 *
546 547 *
547 548 *
548 549 *
549 550 *
550 551 *
551 552 *
552 553 *
553 554 *
554 555 *
555 556 *
556 557 *
557 558 *
558 559 *
559 560 *
560 561 *
561 562 *
562 563 *
563 564 *
564 565 *
565 566 *
566 567 *
567 568 *
568 569 *
569 570 *
570 571 *
571 572 *
572 573 *
573 574 *
574 575 *
575 576 *
576 577 *
577 578 *
578 579 *
579 580 *
580 581 *
581 582 *
582 583 *
583 584 *
584 585 *
585 586 *
586 587 *
587 588 *
588 589 *
589 590 *
590 591 *
591 592 *
592 593 *
593 594 *
594 595 *
595 596 *
596 597 *
597 598 *
598 599 *
599 600 *
600 601 *
601 602 *
602 603 *
603 604 *
604 605 *
605 606 *
606 607 *
607 608 *
608 609 *
609 610 *
610 611 *
611 612 *
612 613 *
613 614 *
614 615 *
615 616 *
616 617 *
617 618 *
618 619 *
619 620 *
620 621 *
621 622 *
622 623 *
623 624 *
624 625 *
625 626 *
626 627 *
627 628 *
628 629 *
629 630 *
630 631 *
631 632 *
632 633 *
633 634 *
634 635 *
635 636 *
636 637 *
637 638 *
638 639 *
639 640 *
640 641 *
641 642 *
642 643 *
643 644 *
644 645 *
645 646 *
646 647 *
647 648 *
648 649 *
649 650 *
650 651 *
651 652 *
652 653 *
653 654 *
654 655 *
655 656 *
656 657 *
657 658 *
658 659 *
659 660 *
660 661 *
661 662 *
662 663 *
663 664 *
664 665 *
665 666 *
666 667 *
667 668 *
668 669 *
669 670 *
670 671 *
671 672 *
672 673 *
673 674 *
674 675 *
675 676 *
676 677 *
677 678 *
678 679 *
679 680 *
680 681 *
681 682 *
682 683 *
683 684 *
684 685 *
685 686 *
686 687 *
687 688 *
688 689 *
689 690 *
690 691 *
691 692 *
692 693 *
693 694 *
694 695 *
695 696 *
696 697 *
697 698 *
698 699 *
699 700 *
700 701 *
701 702 *
702 703 *
703 704 *
704 705 *
705 706 *
706 707 *
707 708 *
708 709 *
709 710 *
710 711 *
711 712 *
712 713 *
713 714 *
714 715 *
715 716 *
716 717 *
717 718 *
718 719 *
719 720 *
720 721 *
721 722 *
722 723 *
723 724 *
724 725 *
725 726 *
726 727 *
727 728 *
728 729 *
729 730 *
730 731 *
731 732 *
732 733 *
733 734 *
734 735 *
735 736 *
736 737 *
737 738 *
738 739 *
739 740 *
740 741 *
741 742 *
742 743 *
743 744 *
744 745 *
745 746 *
746 747 *
747 748 *
748 749 *
749 750 *
750 751 *
751 752 *
752 753 *
753 754 *
754 755 *
755 756 *
756 757 *
757 758 *
758 759 *
759 760 *
760 761 *
761 762 *
762 763 *
763 764 *
764 765 *
765 766 *
766 767 *
767 768 *
768 769 *
769 770 *
770 771 *
771 772 *
772 773 *
773 774 *
774 775 *
775 776 *
776 777 *
777 778 *
778 779 *
779 780 *
780 781 *
781 782 *
782 783 *
783 784 *
784 785 *
785 786 *
786 787 *
787 788 *
788 789 *
789 790 *
790 791 *
791 792 *
792 793 *
793 794 *
794 795 *
795 796 *
796 797 *
797 798 *
798 799 *
799 800 *
800 801 *
801 802 *
802 803 *
803 804 *
804 805 *
805 806 *
806 807 *
807 808 *
808 809 *
809 810 *
810 811 *
811 812 *
812 813 *
813 814 *
814 815 *
815 816 *
816 817 *
817 818 *
818 819 *
819 820 *
820 821 *
821 822 *
822 823 *
823 824 *
824 825 *
825 826 *
826 827 *
827 828 *
828 829 *
829 830 *
830 831 *
831 832 *
832 833 *
833 834 *
834 835 *
835 836 *
836 837 *
837 838 *
838 839 *
839 840 *
840 841 *
841 842 *
842 843 *
843 844 *
844 845 *
845 846 *
846 847 *
847 848 *
848 849 *
849 850 *
850 851 *
851 852 *
852 853 *
853 854 *
854 855 *
855 856 *
856 857 *
857 858 *
858 859 *
859 860 *
860 861 *
861 862 *
862 863 *
863 864 *
864 865 *
865 866 *
866 867 *
867 868 *
868 869 *
869 870 *
870 871 *
871 872 *
872 873 *
873 874 *
874 875 *
875 876 *
876 877 *
877 878 *
878 879 *
879 880 *
880 881 *
881 882 *
882 883 *
883 884 *
884 885 *
885 886 *
886 887 *
887 888 *
888 889 *
889 890 *
890 891 *
891 892 *
892 893 *
893 894 *
894 895 *
895 896 *
896 897 *
897 898 *
898 899 *
899 900 *
900 901 *
901 902 *
902 903 *
903 904 *
904 905 *
905 906 *
906 907 *
907 908 *
908 909 *
909 910 *
910 911 *
911 912 *
912 913 *
913 914 *
914 915 *
915 916 *
916 917 *
917 918 *
918 919 *
919 920 *
920 921 *
921 922 *
922 923 *
923 924 *
924 925 *
925 926 *
926 927 *
927 928 *
928 929 *
929 930 *
930 931 *
931 932 *
932 933 *
933 934 *
934 935 *
935 936 *
936 937 *
937 938 *
938 939 *
939 940 *
940 941 *
941 942 *
942 943 *
943 944 *
944 945 *
945 946 *
946 947 *
947 948 *
948 949 *
949 950 *
950 951 *
951 952 *
952 953 *
953 954 *
954 955 *
955 956 *
956 957 *
957 958 *
958 959 *
959 960 *
960 961 *
961 962 *
962 963 *
963 964 *
964 965 *
965 966 *
966 967 *
967 968 *
968 969 *
969 970 *
970 971 *
971 972 *
972 973 *
973 974 *
974 975 *
975 976 *
976 977 *
977 978 *
978 979 *
979 980 *
980 981 *
981 982 *
982 983 *
983 984 *
984 985 *
985 986 *
986 987 *
987 988 *
988 989 *
989 990 *
990 991 *
991 992 *
992 993 *
993 994 *
994 995 *
995 996 *
996 997 *
997 998 *
998 999 *
999 1000 *
1000 1001 *
1001 1002 *
1002 1003 *
1003 1004 *
1004 1005 *
1005 1006 *
1006 1007 *
1007 1008 *
1008 1009 *
1009 1010 *
1010 1011 *
1011 1012 *
1012 1013 *
1013 1014 *
1014 1015 *
1015 1016 *
1016 1017 *
1017 1018 *
1018 1019 *
1019 1020 *
1020 1021 *
1021 1022 *
1022 1023 *
1023 1024 *
1024 1025 *
1025 1026 *
1026 1027 *
1027 1028 *
1028 1029 *
1029 1030 *
1030 1031 *
1031 1032 *
1032 1033 *
1033 1034 *
1034 1035 *
1035 1036 *
1036 1037 *
1037 1038 *
1038 1039 *
1039 1040 *
1040 1041 *
1041 1042 *
1042 1043 *
1043 1044 *
1044 1045 *
1045 1046 *
1046 1047 *
1047 1048 *
1048 1049 *
1049 1050 *
1050 1051 *
1051 1052 *
1052 1053 *
1053 1054 *
1054 1055 *
1055 1056 *
1056 1057 *
1057 1058 *
1058 1059 *
1059 1060 *
1060 1061 *
1061 1062 *
1062 1063 *
1063 1064 *
1064 1065 *
1065 1066 *
1066 1067 *
1067 1068 *
1068 1069 *
1069 1070 *
1070 1071 *
1071 1072 *
1072 1073 *
1073 1074 *
1074 1075 *
1075 1076 *
1076 1077 *
1077 1078 *
1078 1079 *
1079 1080 *
1080 1081 *
1081 1082 *
1082 1083 *
1083 1084 *
1084 1085 *
1085 1086 *
1086 1087 *
1087 1088 *
1088 1089 *
1089 1090 *
1090 1091 *
1091 1092 *
1092 1093 *
1093 1094 *
1094 1095 *
1095 1096 *
1096 1097 *
1097 1098 *
1098 1099 *
1099 1100 *
1100 1101 *
1101 1102 *
1102 1103 *
1103 1104 *
1104 1105 *
1105 1106 *
1106 1107 *
1107 1108 *
1108 1109 *
1109 1110 *
1110 1111 *
1111 1112 *
1112 1113 *
1113 1114 *
1114 1115 *
1115 1116 *
1116 1117 *
1117 1118 *
1118 1119 *
1119 1120 *
1120 1121 *
1121 1122 *
1122 1123 *
1123 1124 *
1124 1125 *
1125 1126 *
1126 1127 *
1127 1128 *
1128 1129 *
1129 1130 *
1130 1131 *
1131 1132 *
1132 1133 *
1133 1134 *
1134 1135 *
1135 1136 *
1136 1137 *
1137 1138 *
1138 1139 *
1139 1140 *
1140 1141 *
1141 1142 *
1142 1143 *
1143 1144 *
1144 1145 *
1145 1146 *
1146 1147 *
1147 1148 *
1148 1149 *
1149 1150 *
1150 1151 *
1151 1152 *
1152 1153 *
1153 1154 *
1154 1155 *
1155 1156 *
1156 1157 *
1157 1158 *
1158 1159 *
1159 1160 *
1160 1161 *
1161 1162 *
1162 1163 *
1163 1164 *
1164 1165 *
1165 1166 *
1166 1167 *
1167 1168 *
1168 1169 *
1169 1170 *
1170 1171 *
1171 1172 *
1172 1173 *
1173 1174 *
1174 1175 *
1175 1176 *
1176 1177 *
1177 1178 *
1178 1179 *
1179 1180 *
1180 1181 *
1181 1182 *
1182 1183 *
1183 1184 *
1184 1185 *
1185 1186 *
1186 1187 *
1187 1188 *
1188 1189 *
1189 1190 *
1190 1191 *
1191 1192 *
1192 1193 *
1193 1194 *
1194 1195 *
1195 1196 *
1196 1197 *
1197 1198 *
1198 1199 *
1199 1200 *
1200 1201 *
1201 1202 *
1202 1203 *
1203 1204 *
1204 1205 *
1205 1206 *
1206 1207 *
1207 1208 *
1208 1209 *
1209 1210 *
1210 1211 *
1211 1212 *
1212 1213 *
1213 1214 *
1214 1215 *
1215 1216 *
1216 1217 *
1217 1218 *
1218 1219 *
1219 1220 *
1220 1221 *
1221 1222 *
1222 1223 *
1223 1224 *
1224 1225 *
1225 1226 *
1226 1227 *
1227 1228 *
1228 1229 *
1229 1230 *
1230 1231 *
1231 1232 *
1232 1233 *
1233 1234 *
1234 1235 *
1235 1236 *
1236 1237 *
1237 1238 *
1238 1239 *
1239 1240 *
1240 1241 *
1241 1242 *
1242 1243 *
1243 1244 *
1244 1245 *
1245 1246 *
1246 1247 *
1247 1248 *
1248 1249 *
1249 1250 *
1250 1251 *
1251 1252 *
1252 1253 *
1253 1254 *
1254 1255 *
1255 1256 *
1256 1257 *
1257 1258 *
1258 1259 *
1259 1260 *
1260 1261 *
1261 1262 *
1262 1263 *
1263 1264 *
1264 1265 *
1265 1266 *
1266 1267 *
1267 1268 *
1268 1269 *
1269 1270 *
1270 1271 *
1271 1272 *
1272 1273 *
1273 1274 *
1274 1275 *
1275 1276 *
1276 1277 *
1277 1278 *
1278 1279 *
1279 1280 *
1280 1281 *
1281 1282 *
1282 1283 *
1283 1284 *
1284 1285 *
1285 1286 *
1286 1287 *
1287 1288 *
1288 1289 *
1289 1290 *
1290 1291 *
1291 1292 *
1292 1293 *
1293 1294 *
1294 1295 *
1295 1296 *
1296 1297 *
1297 1298 *
1298 1299 *
1299 1300 *
1300 1301 *
1301 1302 *
1302 1303 *
1303 1304 *
1304 1305 *
1305 1306 *
1306 1307 *
1307 1308 *
1308 1309 *
1309 1310 *
1310 1311 *
1311 1312
```

Program TA.C

```
1  /* ta.c */
2
3  /* #defines set on compilation command line:
4  **      K_F      Keil/Franklin
5  **      I_A      IAR/Archimedes
6  **      -----
7  **      LARGE    large memory model
8  **      COMPACT  compact memory model
9  **      SMALL    small memory model
10 **      -----
11 **      A51      8051 architecture
12 **      A11      68HC11 architecture
13 */
14
15 #ifdef K_F      /*{*/
16 # pragma pl(86) /* Listing File Page Length (lines) */
17 # pragma pw(132) /* Listing File Page Width (characters) */
18 #endif /*K_F*/ /*}*/
19
20 #include <stdio.h>
21 /*#include <stdarg.h>*/
22 #include <string.h>
23
24 #if defined(LARGE) /*{ large memory model */
25 # define ARYSIZ 10
26 #elif defined(COMPACT) /*{ compact memory model */
27 # define ARYSIZ 7
28 #elif defined(SMALL) /*{ small memory model */
29 # define ARYSIZ 4
30 #else
31 ??? /*(force syntax error)*/
32 #endif /*}} memory model */
33
34 # define LIST_SIZ 3
35 # define TRUE 1
36
37 /*****/
38 /* typedefs */
39 /*****/
40 typedef unsigned char uchar;
41 typedef unsigned int uint;
42
43 #if defined(K_F) /*{*/
44     typedef struct List {
45         uint list_uival;
46         struct List* list_p_nxt;
47     } LIST;
48     typedef struct List* P_LIST;
49 #elif defined(I_A) /*}{*/
50     typedef struct List* P_LIST;
51     typedef struct List {
52         uint list_uival;
53         P_LIST list_p_nxt;
54     } LIST;
55 #endif /*compiler*/ /*}}*/
```

```

56
57             /*******/
58             /* main() */
59             /*******/
60     main ()
61     {
62         float      iteration = 0.0;
63         uchar      ary[ARYSIZ];
64         uint       ai = 1, aj = 2;
65         LIST       nodes[LIST_SIZ];
66         uint       li, lj;
67         P_LIST     list_head = (P_LIST)NULL;
68         P_LIST     p_list;
69         uint       main_loop_factor = 1;
70
71         while (TRUE) {
72
73             Array_Init: /* (for demo's -- in case lines inserted/deleted) */
74                 /* array demo */
75                 for ( ai = 0; ai < ARYSIZ; ai++ ) {
76                     aj = (ai * 2) + main_loop_factor;
77                     ary[ai] = aj;
78                 } /*for ai*/
79
80             List_Init: /* (for demo's -- in case lines inserted/deleted) */
81                 /* structure/list demo */
82                 /* initialize fields/links */
83                 for ( li = 0; li < LIST_SIZ; li++ ) {
84                     nodes[li].list_uival = (li * 4) + main_loop_factor;
85                     nodes[li].list_p_nxt = (li == 0) ? NULL : &(nodes[li-1]);
86                 } /*for li*/
87                 list_head = &(nodes[LIST_SIZ-1]);
88
89             List_Scan: /* (for demo's -- in case lines inserted/deleted) */
90                 /* scan through nodes in list */
91                 p_list = list_head;
92                 while (p_list != NULL) {
93                     lj = p_list->list_uival;
94                     p_list = p_list->list_p_nxt;
95                 } /*while p_list*/
96
97                 iteration += 1.1;
98                 main_loop_factor++;
99
100             End_Loop: ; /* (for demo's -- in case lines inserted/deleted) */
101
102                 } /*while TRUE*/
103
104         } /*main*/

```

Watch Window

The program has two parts. The first part is a loop to initialize the elements in an array of **unsigned chars**. The second part uses an array of **structures** to implement a singly-linked list. The first sub-loop in this second part establishes the linkage through the nodes in the list. The second sub-loop traverses the linked list using a pointer variable.

We have already loaded the program TAKF51L.AOM into the *iceMASTER* emulator (TAKF51L.AOM: Test/program A, Keil/Franklin compiler, MCS-51 chip architecture, Large memory model, Absolute Object Module format). We have also opened the Watch Window and set several Watch Expressions:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Module:TA		Source	
File:takf51l.aom			
75	for (ai = 0; ai < ARYSIZ; ai++) {		
76	aj = (ai * 2) + main_loop_factor;		
77	ary[ai] = aj;		
78	} /*for ai*/		
Watch			
X:008E uint	ai	1	
X:0090 uint	aj	2	
X:0085 uchar	ary[ai]	0xFF '.'	255
X:0084 [10]uchar	ary	[10]uchar	
X:0084 uchar	[0] 0xFF '.'	255
X:0085 uchar	[1] 0xFF '.'	255
X:0086 uchar	[2] 0xFF '.'	255
X:0087 uchar	[3] 0xFF '.'	255
X:0088 uchar	[4] 0xFF '.'	255
X:0089 uchar	[5] 0xFF '.'	255
X:008A uchar	[6] 0xFF '.'	255
X:008B uchar	[7] 0xFF '.'	255
X:008C uchar	[8] 0xFF '.'	255
X:008D uchar	[9] 0xFF '.'	255

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTr4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr2StepTo

The program is about to step through the array *ary* to initialize each element. There are four Watch Expressions in the Watch Window. *ai* and *aj* are simple **unsigned integer** variables at locations 0x008E and 0x0090, respectively, in External Data memory. Their current values are 1 and 2 (from source line #64). The third Watch Expression, *ary[ai]* monitors a single element in the array *ary*, but the element being monitored changes as the value of the subscript *ai* changes. The fourth Watch Expression, *ary*, watches the entire array. The "value" shown for *ary* is its data type: "[10]uchar", meaning that it is an array of 10 elements, each of which is an **unsigned char**. Following this header line for the array, each element in the array appears. The current value in each element is displayed in hexadecimal, character and unsigned decimal formats, preceded by the element's actual subscript.

Let's single-step the program by one HLL (Higher-Level Language) source line:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Module:TA		Source File:takf511.aom	
75	for (ai = 0; ai < ARYSIZ; ai++) {		
76	aj = (ai * 2) + main_loop_factor;		
77	ary[ail] = aj;		
78	} /*for ai*/		
Watch			
X:008E uint	ai	3	
X:0090 uint	aj	2	
X:0084 uchar	ary[ail]	0xFF ' ' 255	
X:0084 [10]uchar	ary	[10]uchar	
X:0084 uchar	[0] 0xFF ' ' 255	
X:0085 uchar	[1] 0xFF ' ' 255	
X:0086 uchar	[2] 0xFF ' ' 255	
X:0087 uchar	[3] 0xFF ' ' 255	
X:0088 uchar	[4] 0xFF ' ' 255	
X:0089 uchar	[5] 0xFF ' ' 255	
X:008A uchar	[6] 0xFF ' ' 255	
X:008B uchar	[7] 0xFF ' ' 255	
X:008C uchar	[8] 0xFF ' ' 255	
X:008D uchar	[9] 0xFF ' ' 255	

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin3StepOvr3StepTo

The program just executed the **for** loop initialization statement ($ai = 0;$) and is about to execute the first line in the **for** loop. Notice that the value of ai has changed from 1 to 0; it is highlighted to notify you that the value changed. Also note that the address for the Watch Expression $ary[ai]$ has changed (because the subscript value ai changed); it, too, is highlighted to flag the change.

Now, let's step twice to get past the first array element assignment at line #77 at the end of the loop:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Module:TA		Source File:takf511.aom	
75	for (ai = 0; ai < ARYSIZ; ai++) {		
76	aj = (ai * 2) + main_loop_factor;		
77	ary[ail] = aj;		
78	} /*for ai*/		
Watch			
X:008E uint	ai	0	
X:0090 uint	aj	1	
X:0084 uchar	ary[ail]	0x01 ' ' 1	
X:0084 [10]uchar	ary	[10]uchar	
X:0084 uchar	[0] 0x01 ' ' 1	
X:0085 uchar	[1] 0xFF ' ' 255	
X:0086 uchar	[2] 0xFF ' ' 255	
X:0087 uchar	[3] 0xFF ' ' 255	
X:0088 uchar	[4] 0xFF ' ' 255	
X:0089 uchar	[5] 0xFF ' ' 255	
X:008A uchar	[6] 0xFF ' ' 255	
X:008B uchar	[7] 0xFF ' ' 255	
X:008C uchar	[8] 0xFF ' ' 255	
X:008D uchar	[9] 0xFF ' ' 255	

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin3StepOvr3StepTo

After executing the assignment $ary[ai] = aj;$, we see that the value stored in the first element (subscript value zero) has changed, and the corresponding change is reflected in the copy being monitored by Watch Expression $ary[ai]$.

Let's take several more trips through the loop:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA		Source				File:takf511.aom	
75	for (ai = 0; ai < ARYSIZ; ai++) {						
76	aj = (ai * 2) + main_loop_factor;						
77	ary[ai] = aj;						
78	} /*for ai*/						
Watch							
X:008E	uint	ai	4				
X:0090	uint	aj	7				
X:0088	uchar	ary[ai]	0xFF '.,' 255				
X:0084	[10]uchar	ary	[10]uchar				
X:0084	uchar		[0]	0x01	'.,'	1	
X:0085	uchar		[1]	0x03	'.,'	3	
X:0086	uchar		[2]	0x05	'.,'	5	
X:0087	uchar		[3]	0x07	'.,'	7	
X:0088	uchar		[4]	0xFF	'.,'	255	
X:0089	uchar		[5]	0xFF	'.,'	255	
X:008A	uchar		[6]	0xFF	'.,'	255	
X:008B	uchar		[7]	0xFF	'.,'	255	
X:008C	uchar		[8]	0xFF	'.,'	255	
X:008D	uchar		[9]	0xFF	'.,'	255	

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTr4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr3StepTo

We're again at the first statement in the loop after making four complete passes through the loop. The first four elements in the array have been initialized and the subscript *ai* is set for the fifth element in the array. Now let's step to the end of the loop one more time:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Module:TA		Source File:takf511.aom	
75	for (ai = 0; ai < ARYSIZ; ai++) {		
76	aj = (ai * 2) + main_loop_factor;		
77	ary[ai] = aj;		
78	} /*for ai*/		
Watch			
X:008E uint	ai	4	
X:0090 uint	aj	9	
X:0088 uchar	ary[ai]	0x09 '.,' 9	
X:0084 [10]uchar	ary	[10]uchar	
X:0084 uchar		[0] 0x01 '.,' 1	
X:0085 uchar		[1] 0x03 '.,' 3	
X:0086 uchar		[2] 0x05 '.,' 5	
X:0087 uchar		[3] 0x07 '.,' 7	
X:0088 uchar		[4] 0x09 '.,' 9	
X:0089 uchar		[5] 0xFF '.,' 255	
X:008A uchar		[6] 0xFF '.,' 255	
X:008B uchar		[7] 0xFF '.,' 255	
X:008C uchar		[8] 0xFF '.,' 255	
X:008D uchar		[9] 0xFF '.,' 255	

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTr4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr3StepTo

Let's move on to the second part of the program.

```

91      p_list = list head;
92      while (p_list != NULL) {
93          lj = p_list->list_uival;
94          p_list = p_list->list_p_nxt;
95      } /*while p_list*/

```

Watch

X:00A8 ->struct List	p_list	? :FFFF{FFFFFF}
C:FFFF?struct List	*p_list	struct List
C:FFFF?uint	list_uival 65282
C:0001?->struct List	list_p_nxt C:CEFF
X:0092 [3]struct List	nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
X:0094 ->struct List	list_p_nxt (null)
X:0097 struct List	[1] struct List
X:0097 uint	list_uival 5
X:0099 ->struct List	list_p_nxt X:0092
X:009C struct List	[2] struct List
X:009C uint	list_uival 9
X:009E ->struct List	list_p_nxt X:0097

Begin emulation

1 Help 2 ResetEm 3 ResetTg 4 Go 5 GoFrom 6 GoUntil 7 StepIns 8 StepLin 9 StepOvr 0 StepTo

At this point, the program has executed the first sub-loop that establishes the list linkage through the **structure** nodes in the array *nodes* (an array of **structures**). The list head pointer, *list_head*, has been set to point at the first node in the list. That node is the last element in the array (subscript value [2], at address 0x009C in External Data memory), as the list was linked "backwards" through the array elements. The list is linked using the *list_p_nxt* field in each node.

The Watch Window contains three Watch Expressions.

The variable *p_list* is a pointer to a LIST node and it currently contains an illegal pointer value. That's fine because the program has not yet assigned a value to *p_list*. A pointer such as *p_list* occupies 3 bytes in memory. The first byte contains a selector value that indicates which memory space (e.g. Code memory, External Data memory, etc.) is being referenced by the rest of the pointer. The other two bytes in the pointer contain the offset into that memory space. In this case, the memory space selector value is invalid (0xFF), so the software displays it's code as a question mark (?) followed by a colon (:) and the offset bytes in hexadecimal. In addition, because the pointer is illegal, all three "raw" bytes in the pointer's value are displayed in hexadecimal within braces.

The second Watch Expression, **p_list*, is much like the Watch Expression *ary[ai]* used in the earlier part of this tutorial. The expression **p_list* designates "the object pointed to by the pointer *p_list*" (C indirection). In this case, the object is a **structure** of type **struct List**. That structure has two members: an unsigned integer *list_uival* and a pointer, *list_p_nxt*, to another instance of the same structure type. Note the question marks (?) between the address and data type name fields for the **structure** denoted by the Watch Expression **p_list*. The host software is flagging this object specially because it had to "follow" an invalid or NULL pointer to get to the object.

The third Watch Expression is *nodes*, which is the array of **structures** containing the nodes in the linked list.

Let's execute the highlighted line to initialize *p_list* so that it points at the first node in the chain:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA				Source		File:takf511.aom	
91		p_list = list_head;					
92		while (p_list != NULL) {					
93		lj = p_list->list_uival;					
94		p_list = p_list->list_p_nxt;					
95		} /*while p_list*/					
Watch							
X:00A8	->struct List	p_list		X:009C	struct List		
X:009C	struct List	*p_list			list_uival 9		
X:009C	uint				list_p_nxt X:0097		
X:009E	->struct List						
X:0092	[3]struct List	nodes		[3]struct List			
X:0092	struct List			[0] struct List			
X:0092	uint			list_uival 1			
X:0094	->struct List			list_p_nxt (null)			
X:0097	struct List			[1] struct List			
X:0097	uint			list_uival 5			
X:0099	->struct List			list_p_nxt X:0092			
X:009C	struct List			[2] struct List			
X:009C	uint			list_uival 9			
X:009E	->struct List			list_p_nxt X:0097			

Begin emulation

1Help 2ResetEm3ResetTg4Go 5GoFrom 5GoUntil7StepIns3StepLin9StepOvr3StepTo

Notice that *p_list* now contains a valid pointer value and that the node denoted by the expression **p_list* is the first node in the linked list. Let's step through to the end of the first pass in this loop:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA				Source		File:takf511.aom	
91		p_list = list_head;					
92		while (p_list != NULL) {					
93		lj = p_list->list_uival;					
94		p_list = p_list->list_p_nxt;					
95		} /*while p_list*/					
Watch							
X:00A8	->struct List	p_list		X:0097	struct List		
X:0097	struct List	*p_list			list_uival 5		
X:0097	uint				list_p_nxt X:0092		
X:0099	->struct List						
X:0092	[3]struct List	nodes		[3]struct List			
X:0092	struct List			[0] struct List			
X:0092	uint			list_uival 1			
X:0094	->struct List			list_p_nxt (null)			
X:0097	struct List			[1] struct List			
X:0097	uint			list_uival 5			
X:0099	->struct List			list_p_nxt X:0092			
X:009C	struct List			[2] struct List			
X:009C	uint			list_uival 9			
X:009E	->struct List			list_p_nxt X:0097			

Begin emulation

1Help 2ResetEm3ResetTg4Go 5GoFrom 5GoUntil7StepIns3StepLin9StepOvr3StepTo

p_list now points at the second node in the chain and the updated Watch Expression **p_list* shows a highlighted copy of that node.

Finally, let's step until very end of the loop:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA			Source			File:takf511.aom	
91	p_list = list_head;						
92	while (p_list != NULL) {						
93	lj = p_list->list_uival;						
94	p_list = p_list->list_p_nxt;						
95	} /*while p_list*/						
Watch							
X:00A8	->struct List	p_list	(null)				
C:0000?	struct List	*p_list	struct List				
C:0000?	uint	list_uival	517			
C:0002?	->struct List	list_p_nxt	?:FFFF{CEFFFF}			
X:0092	[3]struct List	nodes	[3]struct List				
X:0092	struct List	[0] struct List				
X:0092	uint	list_uival	1			
X:0094	->struct List	list_p_nxt	(null)			
X:0097	struct List	[1] struct List				
X:0097	uint	list_uival	5			
X:0099	->struct List	list_p_nxt	X:0092			
X:009C	struct List	[2] struct List				
X:009C	uint	list_uival	9			
X:009E	->struct List	list_p_nxt	X:0097			

Begin emulation

1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr3StepTo

Here we see that *p_list* is NULL, as expected. Also note that the object denoted by the expression **p_list* is again flagged as "bad" because the *iceMASTER* host software followed a NULL pointer to get to it.

Browsing

You can "jump into" the Watch Window to browse/inspect/change the values in objects designated by the Watch Expressions. To do this, select the *Configure|Windows|Goto* command (shortcut: press the **Tab** key). As you scroll up or down, if the currently highlighted item is a scalar (integral or floating-point) or a pointer, you will see the **Enter:Change** prompt appear on the top-left border of the Watch Window. When you press the **Enter** key, a Change Box pops up to allow you to change the value stored in the highlighted item. A Change Box shows the address, size and data type of the object, as well as its current value. In this case, we are given the opportunity to change a member (*list_uival*) in a structure (which is itself an element in an array):

Module:TA	Source	File:takf511.aom
91	p_list = list_head;	
92	while (p_list != NULL) {	
93	lj = p_list->list_uival;	
94	p_list = p_list->list_p_nxt;	
95	} /*while p_list*/	
Enter:Change		Watch
X:00A8 ->struct List	p_list	(null)
C:0000?struct List	*p_list	struct List
C:0000?uint	list_uival 517
C:0002?->struct List	list_p_nxt ?::FFFF{CEFFFF}
X:0092 [3]struct List	nodes	[3]struct List
X:0092 struct List	[0] struct List
X:0092 uint	list_uival 1
Change Value (Content)		
Addr: X:0092	Size: 2 bytes	Type: uint
Content		list_p_nxt (null)
Current: 1		1] struct List
New:		list_uival 5
		list_p_nxt X:0092
Esc: Do not change current value		2] struct List
		list_uival 9
X:009E ->struct List	list_p_nxt X:0097
Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit		
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo		

If the currently highlighted item is a pointer, structure, union or array, you will see the **Ctrl-B:Browse** prompt appear on the top-right border of the Watch Window. This means that you can "browse" (inspect and/or change) the designated object. Here we'll press **Ctrl-B** to pop up a Browse Window to look at the node pointed at by the link field (*list_p_nxt*) in the first node in the list (that is, we'll "follow" the pointer):

Module: TA	Source	File: takf511.aom
91	p_list = list_head;	
92	while (p_list != NULL) {	
93	lj = p_list->list_uival;	
94	p_list = p_list->list_p_nxt;	
95	} /*while p_list*/	
Enter:Change Watch Ctrl-B:Browse		
X:00A8	->struct List p_list (null)	
C:0000	X:009E ->struct List	t List
C:0000	Addr Data Type Value	t_uival 517
C:0002		t_p_nxt ? :FFFF{CEFFFF}
X:0092	X:0097 struct List struct List	ruct List
X:0092	X:0097 uint list_uival 5	struct List
X:0092	X:0099 ->struct List list_p_nxt X:0092	list_uival 1
X:0094		list_p_nxt (null)
X:0097	struct List	[1] struct List
X:0097	uint	list_uival 5
X:0099	->struct List	list_p_nxt X:0092
X:009C	struct List	[2] struct List
X:009C	uint	list_uival 9
X:009E	->struct List	list_p_nxt X:0097
Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit		
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr2StepTo		

A Browse Window shows the entire object denoted by the pointer. A Browse Window is scrollable, moveable and resizeable. Thus, you can inspect large objects with no problem. You can even write the entire contents of a Browse Window to a file of your choice. Let's move the highlight bar to the link field (*list_p_nxt*) in the struct we are inspecting:

Module: TA	Source	File: takf511.aom
91	p_list = list_head;	
92	while (p_list != NULL) {	
93	lj = p_list->list_uival;	
94	p_list = p_list->list_p_nxt;	
95	} /*while p_list*/	
Enter:Change Watch Ctrl-B:Browse		
X:00A8	->struct List p_list (null)	
C:0000	X:009E ->struct List	t List
C:0000	Addr Data Type Value	t_uival 517
C:0002		t_p_nxt ? :FFFF{CEFFFF}
X:0092	X:0097 struct List struct List	ruct List
X:0092	X:0097 uint list_uival 5	struct List
X:0092	X:0099 ->struct List list_p_nxt X:0092	list_uival 1
X:0094	Enter:Change Ctrl-B:Browse	list_p_nxt (null)
X:0097	struct List	[1] struct List
X:0097	uint	list_uival 5
X:0099	->struct List	list_p_nxt X:0092
X:009C	struct List	[2] struct List
X:009C	uint	list_uival 9
X:009E	->struct List	list_p_nxt X:0097
Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit		
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr2StepTo		

Notice that the bottom border of the Browse Window shows that we can change this pointer value (**Enter:Change** on the left) or we can follow it (**Ctrl-B:Browse** on the right). Let's follow it to the next node in the list.

```

Module: TA                                     Source                                     File: takf511.aom
91      p_list = list_head;
92      while (p_list != NULL) {
93          lj = p_list->list_uival;
94          p_list = p_list->list_p_nxt;
95      } /*while p_list*/
Enter: Change                                     Watch                                     Ctrl-B: Browse
X:00A8 ->struct List p_list (null)
C:0000 X:009E ->struct List t List
C:0000 Addr X:0099 ->struct List 517
C:0002 Addr Data Type Value
X:0092 X:0097 X:0092 struct List struct List
X:0092 X:0097 X:0092 uint list_uival 1
X:0094 ->struct List list_p_nxt (null)
X:0097 struct Change Value (Content)
X:0097 uint Addr: X:0094 Size: 3 bytes Type: ->struct List
X:0099 ->struc Content
X:009C struct Current: (null)
X:009C uint New: X:0x009C
X:009E ->struc Esc: Do not change current value

```

Tab/Shift-Tab: Next/Prev window (Keypad): Scroll Esc: Exit
1 Help 2 ResetEm3 ResetTg4 Go 5 GoFrom 6 GoUntil7 StepIns8 StepLin9 StepOvr0 StepTo

Here we see that we are at the last node in the linked list (the link field `list_p_nxt` is NULL). Let's move to that field and change the link to make the list circular. We'll set the link field to point at location 0x009C in External Data memory. That is the location of the head (first) node in the chain.

Now if we continue to follow the link field (repetitively pressing **Ctrl-B** while the highlight bar is on the member `list_p_nxt`), we'll go on, and on, and on ...

```

Module: TA                                     Source                                     File: takf511.aom
91      p_list = list_head;
92      while (p_list != NULL) {
93          lj = p_list->list_uival;
94          p_list = p_list->list_p_nxt;
95      } /*while p_list*/
Enter: Change                                     Watch                                     Ctrl-B: Browse
X:00A8 ->struct List p_list (null)
C:0000 X:009E ->struct List t List
C:0000 Addr X:0099 ->struct List 517
C:0002 Addr X:0094 ->struct List {CEFFFF}
X:0092 X:0097 X:0092 X:009E ->struct List
X:0092 X:0097 X:0092 X:0099 ->struct List
X:0094 ->struct List Data Type Value
Addr X:009E ->struct List
Addr X:0099 ->struct List struct List
X:009C Addr Data Type Value list_uival 1
X:009C X:0097 list_p_nxt X:009C
X:009E X:0097 X:0092 struct List struct List Ctrl-B: Browse
X:009E X:0097 X:0092 uint list_uival 1 t uival 9
Enter X:0099 X:0092 uint list_p_nxt X:009C t_p_nxt X:0097
X:0099 ->struct List

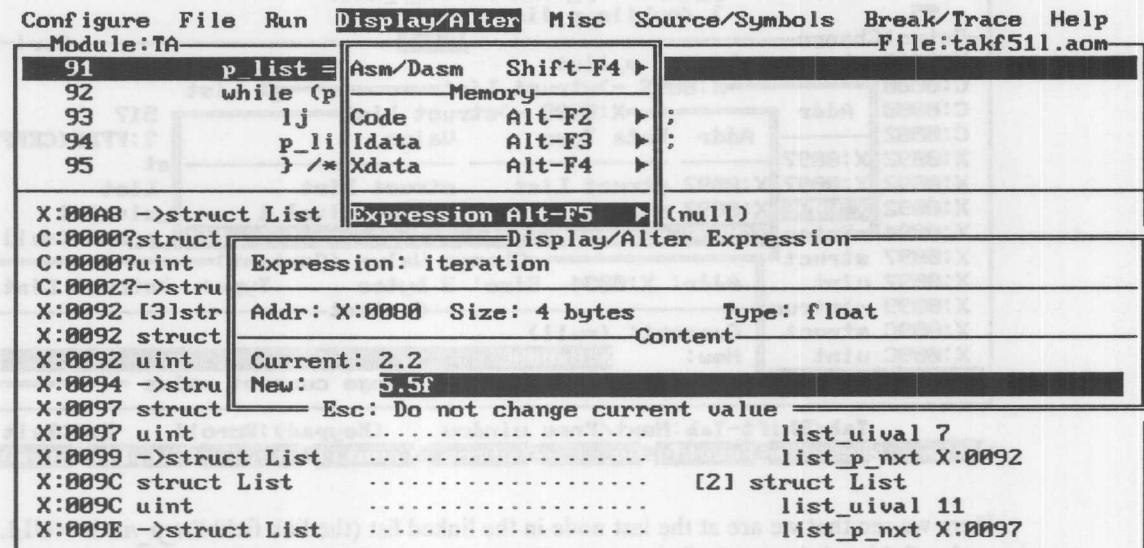
```

Tab/Shift-Tab: Next/Prev window (Keypad): Scroll Esc: Exit
1 Help 2 ResetEm3 ResetTg4 Go 5 GoFrom 6 GoUntil7 StepIns8 StepLin9 StepOvr0 StepTo

Enough.

Display/Alter | Expression Command

The *Display/Alter|Expression* command pops up a dialog box which allows you to view or change the value stored in a location designated by an arbitrary expression:

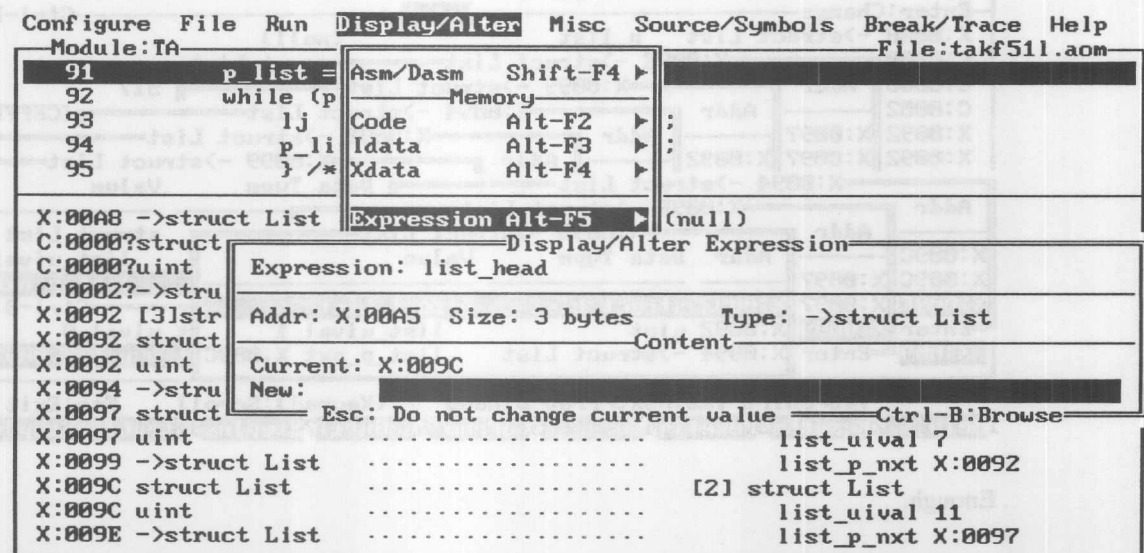


View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo

The dialog box shows the address, size (in bytes or bits) and data type of the object denoted by the expression you type (in this case, the variable *iteration*). Additionally, it shows the current value of the expression (2.2 in this case), and gives you the opportunity to assign a new value to the location designated by the expression. Here we are changing the value/content of *iteration* from 2.2 to 5.5).

If the expression has type "pointer to ...", you can either change the pointer's value by entering a new value or follow the pointer (the **Ctrl-B:Browse** prompt appears on the bottom-right border of the box):



View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

If we choose to follow it by pressing Ctrl-B, again a Browse Window pops up to let us do just that:

Configure File Run **Display/Alter** Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

91	p_list =	Asm/Dasm	Shift-F4 ▶
92	while (p	Memory	
93	lj	Code	Alt-F2 ▶ ;
94	p_li	Idata	Alt-F3 ▶ ;
95	}/*	Xdata	Alt-F4 ▶

X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct Display/Alter Expression
C:0000?uint Expression: list_head
C:0002?->stru X:00A5 ->struct List list_head

Addr	Data	Type	Value
X:009C	struct List	struct List	
X:009C	uint	list_uival 11	
X:009E	->struct List	list_p_next X:0097	
X:0097	uint	Ctrl-R:Resize Ctrl-U:Move	
X:0099	->struct List	list_p_next X:0092	
X:009C	struct List	[2] struct List	
X:009C	uint	list_uival 11	
X:009E	->struct List	list_p_next X:0097	

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr3StepTo

If the expression has type "array of ...", the *Display/Alter|Expression* command shows us the size of the entire array *nodes* (15 bytes in this) case and prompts us that we can browse (inspect) the array (Ctrl-B:Browse prompt on the bottom-right border of the box):

Configure File Run **Display/Alter** Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

91	p_list =	Asm/Dasm	Shift-F4 ▶
92	while (p	Memory	
93	lj	Code	Alt-F2 ▶ ;
94	p_li	Idata	Alt-F3 ▶ ;
95	}/*	Xdata	Alt-F4 ▶

X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct Display/Alter Expression
C:0000?uint Expression: nodes
C:0002?->stru Addr: X:0092 Size: 15 bytes Type: [3]struct List

"<expr>" or "<expr> <new value>"		Ctrl-B:Browse
X:0097	uint	list_uival 7
X:0099	->struct List	list_p_next X:0092
X:009C	struct List	[2] struct List
X:009C	uint	list_uival 11
X:009E	->struct List	list_p_next X:0097

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr3StepTo

Pressing Ctrl-B pops-up a Browse Window, letting us inspect and/or change the individual elements in the array:

Configure File Run **Display/Alter** Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

91	p_list =	Asm/Dasm	Shift-F4 ▶	
92	while (p	Memory		
93	lj	Code	Alt-F2 ▶	;
94	p_li	Idata	Alt-F3 ▶	;
95	} /*	Xdata	Alt-F4 ▶	

X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct X:0092 [3]struct List nodes
C:0000?uint E Addr Data Type Value
C:0002?->stru A X:0092 [3]struct List [3]struct List
X:0092 [3]str X:0092 struct List [0] struct List
X:0092 struct X:0092 uint list_uival 3
X:0092 uint X:0094 ->struct List list_p_nxt (null)
X:0094 ->stru X:0097 struct List [1] struct List
X:0097 struct X:0097 uint list_uival 7
X:0097 uint X:0099 ->struct List list_p_nxt X:0092
X:0099 ->struct X:009C struct List [2] struct List
X:009C struct Li X:009C uint list_uival 11
X:009C uint X:009E ->struct List list_p_nxt X:0097
X:009E ->struct

Ctrl-R:Resize Ctrl-V:Move

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo

You can enter almost any expression in the *Display/Alter|Expression* dialog box, designating array elements and struct/union members just as you would in your C program:

Configure File Run **Display/Alter** Misc Source/Symbols Break/Trace Help
Module:TA File:takf511.aom

91	p_list =	Asm/Dasm	Shift-F4 ▶	
92	while (p	Memory		
93	lj	Code	Alt-F2 ▶	;
94	p_li	Idata	Alt-F3 ▶	;
95	} /*	Xdata	Alt-F4 ▶	

X:00A8 ->struct List Expression Alt-F5 ▶ (null)
C:0000?struct Display/Alter Expression
C:0000?uint Expression: nodes[1].list_p_nxt->list_uival
C:0002?->stru
X:0092 [3]str Addr: X:0092 Size: 2 bytes Type: uint
X:0092 struct Content
X:0092 uint Current: 3
X:0094 ->stru New:
X:0097 struct Esc: Do not change current value
X:0097 uint list_uival 7
X:0099 ->struct List list_p_nxt X:0092
X:009C struct List [2] struct List
X:009C uint list_uival 11
X:009E ->struct List list_p_nxt X:0097

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo

Or you can follow a pointer through several levels of indirection in one step:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA							
91	p_list =	Asm/Dasm	Shift-F4				
92	while (p	Memory					
93	lj	Code	Alt-F2	;			
94	p_li	Idata	Alt-F3	;			
95	}/*	Xdata	Alt-F4				
X:00A8 ->struct List		Expression Alt-F5		(null)			
C:0000?struct		Display/Alter Expression					
C:0000?uint		Expression: list_head->list_p_nxt->list_p_nxt					
C:0002?->stru							
X:0092 [3]str		Addr: X:0099		Size: 3 bytes		Type: ->struct List	
X:0092 struct		Content					
X:0092 uint		Current: X:0092					
X:0094 ->stru		New:					
X:0097 struct		Esc: Do not change current value Ctrl-B:Browse					
X:0097 uint		list_uival 7					
X:0099 ->struct List		list_p_nxt X:0092					
X:009C struct List		[2] struct List					
X:009C uint		list_uival 11					
X:009E ->struct List		list_p_nxt X:0097					

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo

And, of course, you can also monitor such complicated expressions in the Watch Window.

Finally, in the simplest case, the *Display/Alter* *Expression* command even operates as a calculator:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA							
91	p_list =	Asm/Dasm	Shift-F4				
92	while (p	Memory					
93	lj	Code	Alt-F2	;			
94	p_li	Idata	Alt-F3	;			
95	}/*	Xdata	Alt-F4				
X:00A8 ->struct List		Expression Alt-F5		(null)			
C:0000?struct		Display/Alter Expression					
C:0000?uint		Expression: 55u+66u					
C:0002?->stru							
X:0092 [3]str		Hex: 0x0079		Size: 2 bytes		Type: uint	
X:0092 struct		Value					
X:0092 uint		Current: 121					
X:0094 ->stru		"<expr>" or "<expr> <new value>"					
X:0097 struct							
X:0097 uint		list_uival 7					
X:0099 ->struct List		list_p_nxt X:0092					
X:009C struct List		[2] struct List					
X:009C uint		list_uival 11					
X:009E ->struct List		list_p_nxt X:0097					

View/change the value in any variable, register, bit, memory location, etc.

1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo

Source/Symbols Commands

The *Source/Symbols* commands display the symbols in your program sorted in various ways. Here we see the local symbols in TAKF51LAOM sorted alphabetically within the module in which they were defined:

Configure					File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA					Source					File:takf51l.aom	
Local Symbols - By Module											
Enter Symbol					(15 Symbols)						
Addr	Symbol Name (M:Module F:Function)		Data Type	Value	Defined Within (#: Scope Level)						
C:00D9	Array_Init		(label)		2 MAIN						
C:027D	End_Loop		(label)		2 MAIN						
C:0135	List_Init		(label)		2 MAIN						
C:01E7	List_Scan		(label)		2 MAIN						
X:008E	ai		uint	10	2 MAIN						
X:0090	aj		uint	21	2 MAIN						
X:0084	ary		[10]uchar	[10]uchar	2 MAIN						
X:0080	iteration		float	5.5	2 MAIN						
X:00A1	li		uint	3	2 MAIN						
X:00A5	list_head		->struct List	X:009C	2 MAIN						
X:00A3	lj		uint	2	2 MAIN						
X:00AB	main_loop_factor		uint	3	2 MAIN						
X:0092	nodes		[3]struct List	[3]struct List	2 MAIN						
X:00A8	p_list		->struct List	(null)	2 MAIN						
Enter:Change Ctrl-R:Resize Ctrl-V:Move											

Source level debug and symbol displays

All of the *Source/Symbols* windows have the same basic format, showing the address of the symbol, the symbol's name, its data type, current value and where it was defined in the program. As you scroll through the window, the **Enter:Change** prompt appears on the bottom-left border of the window if you can change the currently highlighted item. This is the case for scalars (integral and floating-point) and pointers. Pressing **Enter** while *li* is highlighted pops up a Change Box to allow you to change its current value:

Configure					File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help	
Module:TA					Source					File:takf51l.aom		
Local Symbols - By Module												
Enter Symbol					(15 Symbols)							
Addr	Symbol Name (M:Module F:Function)			Data Type	Value		Defined Within (#: Scope Level)					
C:00D9	Array_Init			(label)			2 MAIN					
C:027D	End_Loop			(label)			2 MAIN					
C:0135	List_Init			(label)			2 MAIN					
C:01E7	List_Scan			(label)			2 MAIN					
X:008E	ai			uint	10		2 MAIN					
X:0090	aj			uint	21		2 MAIN					
X:0084	ary			[10]uchar	[10]uchar		2 MAIN					
X:0080	iteration			float	5.5		2 MAIN					
X:00A1	li			uint	3		2 MAIN					
X:	Change Value (Content)						X:009C	2 MAIN				
X:	Addr: X:00A1		Size: 2 bytes		Type: uint		2		2 MAIN			
X:	Content						3		2 MAIN			
X:	Current: 3				ruct List		2		2 MAIN			
X:	New:				(null)		2		2 MAIN			
E	Esc: Do not change current value											e

Source level debug and symbol displays

If a symbol is an array, structure or union, the "value" shown in the *Source/Symbols* windows is the symbol's data type:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help				
Module:TA Source File:takf511.aom				
Local Symbols - By Module				
Enter Symbol		(15 Symbols)		
Addr	Symbol Name (M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:00D9	Array_Init	(label)		2 MAIN
C:027D	End_Loop	(label)		2 MAIN
C:0135	List_Init	(label)		2 MAIN
C:01E7	List_Scan	(label)		2 MAIN
X:008E	ai	uint	10	2 MAIN
X:0090	aj	uint	21	2 MAIN
X:0084	ary	[10]uchar	[10]uchar	2 MAIN
X:0080	iteration	float	5.5	2 MAIN
X:00A1	li	uint	3	2 MAIN
X:00A5	list_head	->struct List	X:009C	2 MAIN
X:00A3	lj	uint	2	2 MAIN
X:00AB	main_loop_factor	uint	3	2 MAIN
X:0092	nodes	[3]struct List	[3]struct List	2 MAIN
X:00A8	p_list	->struct List	(null)	2 MAIN
Ctrl-R:Resize Ctrl-U:Move Ctrl-B:Browse				

Source level debug and symbol displays

Notice that the **Ctrl-B:Browse** prompt appears on the bottom-right border of the window. You can browse (inspect) structures, unions, arrays and pointers in all the *Source/Symbols* windows. Pressing **Ctrl-B** here pops up a Browse Window, allowing us to inspect the individual elements in the array *ary*:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help				
Module:TA		Source		File:takf511.aom
Local Symbols - By Module				
X:0084 [10]uchar ary			Ctrl-W:Write	(15 Symbols)
Addr	Data Type	Value	Defined Within #: Scope Level)	
X:0084	[10]uchar	[10]uchar		
X:0084	uchar	[0] 0x03 ' ' 3		
X:0085	uchar	[1] 0x05 ' ' 5	MAIN	
X:0086	uchar	[2] 0x07 ' ' 7	MAIN	
X:0087	uchar	[3] 0x09 ' ' 9	MAIN	
X:0088	uchar	[4] 0x0B ' ' 11	MAIN	
X:0089	uchar	[5] 0x0D ' ' 13	MAIN	
X:008A	uchar	[6] 0x0F ' ' 15	MAIN	
X:008B	uchar	[7] 0x11 ' ' 17	MAIN	
X:008C	uchar	[8] 0x13 ' ' 19	MAIN	
X:008D	uchar	[9] 0x15 ' ' 21	MAIN	
Enter:Change Ctrl-R:Resize Ctrl-U:Move				
X:00A3	lj	uint	2 2	MAIN
X:00AB	main_loop_factor	uint	3 2	MAIN
X:0092	nodes	[3]struct List	[3]struct List	2 MAIN
X:00A8	p_list	->struct List	(null)	2 MAIN
Ctrl-R:Resize Ctrl-U:Move Ctrl-B:Browse				

Source level debug and symbol displays

Similarly, if the currently-highlighted symbol is a pointer to something, we can press the **Ctrl-B** key to browse that "something". And, of course, if that "something" contains elements that are browseable, we can inspect or change them, too:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help									
Module:TA					Source File:takf51l.aom				
Local Symbols - By Module									
X:00A5 ->struct List list_head					(15 Symbols)				
X:009E ->struct List									
X:0099 ->struct List					Defined Within				
					(#: Scope Level)				
Addr	Addr	Addr	Data Type	Value					
X:009C	X:0097	X:0092	struct List	struct List	0				
X:009C	X:0097	X:0092	uint	list_uival 3	7	1	TA		
X:009E	X:0097	X:0094	->struct List	list_p_nxt (null)					
Enter	Enter:Change	Ctrl-B:Browse							
C:00D9	Arr				2	MAIN			
C:027D	End_Loop	(label)			2	MAIN			
C:0135	List_Init	(label)			2	MAIN			
C:01E7	List_Scan	(label)			2	MAIN			
X:008E	ai	uint			10	2	MAIN		
X:0090	aj	uint			21	2	MAIN		
X:0084	ary	[10]uchar			[10]uchar	2	MAIN		
X:0080	iteration	float			5.5	2	MAIN		
X:00A1	li	uint			3	2	MAIN		
X:00A5	list_head	->struct List			X:009C	2	MAIN		
X:00A3	lj	uint			2	2	MAIN		
Enter:Change					Ctrl-R:Resize Ctrl-V:Move				
					Ctrl-B:Browse				

Source level debug and symbol displays

Notice that the Ctrl-Browse prompt appears on the bottom-right border of the window. You can browse (inspect) structures, unions, arrays, and pointers in all the source windows. Pressing Ctrl-B here pops up a Browse Window, allowing us to inspect the individual elements in the array.

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Module: TA

Source

Local Symbols - By Module

(15 Symbols)

Addr	Addr	Addr	Data Type	Value	Defined Within (#: Scope Level)
X:0084	X:0084	X:0084	[10]uchar	[10]uchar	2 MAIN
X:0084	X:0084	X:0084	uchar	1 01 0x01	2 MAIN
X:0084	X:0084	X:0084	uchar	1 02 0x02	2 MAIN
X:0084	X:0084	X:0084	uchar	1 03 0x03	2 MAIN
X:0084	X:0084	X:0084	uchar	1 04 0x04	2 MAIN
X:0084	X:0084	X:0084	uchar	1 05 0x05	2 MAIN
X:0084	X:0084	X:0084	uchar	1 06 0x06	2 MAIN
X:0084	X:0084	X:0084	uchar	1 07 0x07	2 MAIN
X:0084	X:0084	X:0084	uchar	1 08 0x08	2 MAIN
X:0084	X:0084	X:0084	uchar	1 09 0x09	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0A 0x0A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0B 0x0B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0C 0x0C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0D 0x0D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0E 0x0E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 0F 0x0F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 10 0x10	2 MAIN
X:0084	X:0084	X:0084	uchar	1 11 0x11	2 MAIN
X:0084	X:0084	X:0084	uchar	1 12 0x12	2 MAIN
X:0084	X:0084	X:0084	uchar	1 13 0x13	2 MAIN
X:0084	X:0084	X:0084	uchar	1 14 0x14	2 MAIN
X:0084	X:0084	X:0084	uchar	1 15 0x15	2 MAIN
X:0084	X:0084	X:0084	uchar	1 16 0x16	2 MAIN
X:0084	X:0084	X:0084	uchar	1 17 0x17	2 MAIN
X:0084	X:0084	X:0084	uchar	1 18 0x18	2 MAIN
X:0084	X:0084	X:0084	uchar	1 19 0x19	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1A 0x1A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1B 0x1B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1C 0x1C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1D 0x1D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1E 0x1E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 1F 0x1F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 20 0x20	2 MAIN
X:0084	X:0084	X:0084	uchar	1 21 0x21	2 MAIN
X:0084	X:0084	X:0084	uchar	1 22 0x22	2 MAIN
X:0084	X:0084	X:0084	uchar	1 23 0x23	2 MAIN
X:0084	X:0084	X:0084	uchar	1 24 0x24	2 MAIN
X:0084	X:0084	X:0084	uchar	1 25 0x25	2 MAIN
X:0084	X:0084	X:0084	uchar	1 26 0x26	2 MAIN
X:0084	X:0084	X:0084	uchar	1 27 0x27	2 MAIN
X:0084	X:0084	X:0084	uchar	1 28 0x28	2 MAIN
X:0084	X:0084	X:0084	uchar	1 29 0x29	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2A 0x2A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2B 0x2B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2C 0x2C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2D 0x2D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2E 0x2E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 2F 0x2F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 30 0x30	2 MAIN
X:0084	X:0084	X:0084	uchar	1 31 0x31	2 MAIN
X:0084	X:0084	X:0084	uchar	1 32 0x32	2 MAIN
X:0084	X:0084	X:0084	uchar	1 33 0x33	2 MAIN
X:0084	X:0084	X:0084	uchar	1 34 0x34	2 MAIN
X:0084	X:0084	X:0084	uchar	1 35 0x35	2 MAIN
X:0084	X:0084	X:0084	uchar	1 36 0x36	2 MAIN
X:0084	X:0084	X:0084	uchar	1 37 0x37	2 MAIN
X:0084	X:0084	X:0084	uchar	1 38 0x38	2 MAIN
X:0084	X:0084	X:0084	uchar	1 39 0x39	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3A 0x3A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3B 0x3B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3C 0x3C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3D 0x3D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3E 0x3E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 3F 0x3F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 40 0x40	2 MAIN
X:0084	X:0084	X:0084	uchar	1 41 0x41	2 MAIN
X:0084	X:0084	X:0084	uchar	1 42 0x42	2 MAIN
X:0084	X:0084	X:0084	uchar	1 43 0x43	2 MAIN
X:0084	X:0084	X:0084	uchar	1 44 0x44	2 MAIN
X:0084	X:0084	X:0084	uchar	1 45 0x45	2 MAIN
X:0084	X:0084	X:0084	uchar	1 46 0x46	2 MAIN
X:0084	X:0084	X:0084	uchar	1 47 0x47	2 MAIN
X:0084	X:0084	X:0084	uchar	1 48 0x48	2 MAIN
X:0084	X:0084	X:0084	uchar	1 49 0x49	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4A 0x4A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4B 0x4B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4C 0x4C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4D 0x4D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4E 0x4E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 4F 0x4F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 50 0x50	2 MAIN
X:0084	X:0084	X:0084	uchar	1 51 0x51	2 MAIN
X:0084	X:0084	X:0084	uchar	1 52 0x52	2 MAIN
X:0084	X:0084	X:0084	uchar	1 53 0x53	2 MAIN
X:0084	X:0084	X:0084	uchar	1 54 0x54	2 MAIN
X:0084	X:0084	X:0084	uchar	1 55 0x55	2 MAIN
X:0084	X:0084	X:0084	uchar	1 56 0x56	2 MAIN
X:0084	X:0084	X:0084	uchar	1 57 0x57	2 MAIN
X:0084	X:0084	X:0084	uchar	1 58 0x58	2 MAIN
X:0084	X:0084	X:0084	uchar	1 59 0x59	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5A 0x5A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5B 0x5B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5C 0x5C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5D 0x5D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5E 0x5E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 5F 0x5F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 60 0x60	2 MAIN
X:0084	X:0084	X:0084	uchar	1 61 0x61	2 MAIN
X:0084	X:0084	X:0084	uchar	1 62 0x62	2 MAIN
X:0084	X:0084	X:0084	uchar	1 63 0x63	2 MAIN
X:0084	X:0084	X:0084	uchar	1 64 0x64	2 MAIN
X:0084	X:0084	X:0084	uchar	1 65 0x65	2 MAIN
X:0084	X:0084	X:0084	uchar	1 66 0x66	2 MAIN
X:0084	X:0084	X:0084	uchar	1 67 0x67	2 MAIN
X:0084	X:0084	X:0084	uchar	1 68 0x68	2 MAIN
X:0084	X:0084	X:0084	uchar	1 69 0x69	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6A 0x6A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6B 0x6B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6C 0x6C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6D 0x6D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6E 0x6E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 6F 0x6F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 70 0x70	2 MAIN
X:0084	X:0084	X:0084	uchar	1 71 0x71	2 MAIN
X:0084	X:0084	X:0084	uchar	1 72 0x72	2 MAIN
X:0084	X:0084	X:0084	uchar	1 73 0x73	2 MAIN
X:0084	X:0084	X:0084	uchar	1 74 0x74	2 MAIN
X:0084	X:0084	X:0084	uchar	1 75 0x75	2 MAIN
X:0084	X:0084	X:0084	uchar	1 76 0x76	2 MAIN
X:0084	X:0084	X:0084	uchar	1 77 0x77	2 MAIN
X:0084	X:0084	X:0084	uchar	1 78 0x78	2 MAIN
X:0084	X:0084	X:0084	uchar	1 79 0x79	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7A 0x7A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7B 0x7B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7C 0x7C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7D 0x7D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7E 0x7E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 7F 0x7F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 80 0x80	2 MAIN
X:0084	X:0084	X:0084	uchar	1 81 0x81	2 MAIN
X:0084	X:0084	X:0084	uchar	1 82 0x82	2 MAIN
X:0084	X:0084	X:0084	uchar	1 83 0x83	2 MAIN
X:0084	X:0084	X:0084	uchar	1 84 0x84	2 MAIN
X:0084	X:0084	X:0084	uchar	1 85 0x85	2 MAIN
X:0084	X:0084	X:0084	uchar	1 86 0x86	2 MAIN
X:0084	X:0084	X:0084	uchar	1 87 0x87	2 MAIN
X:0084	X:0084	X:0084	uchar	1 88 0x88	2 MAIN
X:0084	X:0084	X:0084	uchar	1 89 0x89	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8A 0x8A	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8B 0x8B	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8C 0x8C	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8D 0x8D	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8E 0x8E	2 MAIN
X:0084	X:0084	X:0084	uchar	1 8F 0x8F	2 MAIN
X:0084	X:0084	X:0084	uchar	1 90 0x90	2 MAIN
X:0084	X:0084	X:0084	uchar	1 91 0x91	2 MAIN
X:0084	X:0084	X:0084	uchar	1 92 0x92	2 MAIN
X:0084	X:0084	X:0084	uchar	1 93 0x93	2 MAIN
X:0084	X:0084	X:0084	uchar	1 94 0x94	2 MAIN
X:0084	X:0084	X:0084	uchar	1 95 0x95	2 MAIN
X:0084	X:0084	X:0084	uchar	1 96 0x96	2 MAIN
X:0084	X:0084	X:0084	uchar	1 97 0x97	2 MAIN

Ctrl-B:Browse Ctrl-V:Move -Parent List (null) 2 MAIN
nodes
main_loop_vector
list
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN
2 2 MAIN

Improved Support for "Unsupported" Programs

Even if your program contains limited data type information, or no data type information at all, you can still benefit from some of the new features in the *iceMASTER* software.

The demo program **DEMO.DBG** is an assembly language program assembled using MetaLink's 8051 cross-assembler. The only debug information output by this assembler (and many other assemblers, too) is symbol name, address and memory space (e.g., Code memory, External Data memory, etc.). There is no data type information.

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
					Source	File: demo.dbg	
0030	900000	START:	MOV	DPTR, #0000H	DPTR=0000h		
0033	C290	OUTERLOOP:	CLR	T2	bit[90h]=1		
0035	75640A		MOV	TEMPCOUNT, #0Ah	direct[64h]=00h		
0038	120050	INNERLOOP:	LCALL	WASTETIME			10050
003B	B290		CPL	T2			
003D	E4		CLR	A			
003E	309001		JNB	T2, SKIPOVER			
0041	F4		CPL	A			
0042	D564F3	SKIPOVER:	DJNZ	TEMPCOUNT, INNERLOOP			
0045	A3		INC	DPTR			
0046	120050		LCALL	WASTETIME			
0049	120050		LCALL	WASTETIME			
004C	80E5		SJMP	OUTERLOOP			
004E	80E0	ENDOFPROGRAM:	SJMP	START			
0050	78FF	WASTETIME:	MOV	R0, #0FFh			
					Watch		
D:0064 (unknown)			TEMPCOUNT		0x0A		

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

In the picture above, we have loaded **DEMO.DBG** into the emulator and added the Watch Expression **TEMPCOUNT** to the Watch Window. Notice that the data type is "(unknown)". In such cases, the host software defaults to displaying the value in hexadecimal only, and the software must be "told" how many bytes to read (via the *Configure|Options|Expressions|Unknown data type size* command). Below, we are about to add a Watch Expression that will cause the host software to treat **TEMPCOUNT** as "true" one-byte unsigned character value.

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
				Source	File:demo.dbg		
Emulator				MOV	DPTR,#0000H	DPTR=0000h	
Mapping			:	CLR	T2	bit[90h]=1	
Code Memory				MOV	TEMPCOUNT,#0Ah	direct[64h]=00h	
Xdata Memory			:	LCALL	WASTETIME	10050	
Mode	#1			CPL	T2		
Display				CLR	A		
Attributes				JNB	T2,SKIPOVER		
Windows				CPL	A		
Modify	Ctrl-F6			DJNZ	TEMPCOUNT,INNERLOOP	10038 direct[64h]=0Ah	
Size	Ctrl-F7			INC	DPTR		
Goto	Ctrl-F8			LCALL	WASTETIME		
Repaint	Alt-F1			LCALL	WASTETIME		
Watch				SJMP	OUTERLOOP		
				SJMP	START		
Add	Ctrl-F9			MOV	R0,#0FFh	R0=64h	
				Add Watch Expression			
D:0064	(uchar)TEMPCOUNT			0A			
Load							

Enter any expression denoting an address.

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

As you can see below, there isn't much difference in the display in this case:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
				Source	File:demo.dbg		
0030	900000	START:	MOV	DPTR,#0000H	DPTR=0000h		
0033	C290	OUTERLOOP:	CLR	T2	bit[90h]=1		
0035	75640A		MOV	TEMPCOUNT,#0Ah	direct[64h]=00h		
0038	120050	INNERLOOP:	LCALL	WASTETIME	10050		
003B	B290		CPL	T2			
003D	E4		CLR	A			
003E	309001		JNB	T2,SKIPOVER			
0041	F4		CPL	A			
0042	D564F3	SKIPOVER:	DJNZ	TEMPCOUNT,INNERLOOP	10038 direct[64h]=0Ah		
0045	A3		INC	DPTR			
0046	120050		LCALL	WASTETIME			
0049	120050		LCALL	WASTETIME			
004C	80E5		SJMP	OUTERLOOP			
004E	80E0	ENDOFFPROGRAM:	SJMP	START			
0050	78FF	WASTETIME:	MOV	R0,#0FFh	R0=64h		
				Watch			
D:0064 (unknown)				TEMPCOUNT	0x0A		
D:0064 uchar				(uchar)TEMPCOUNT	0x0A '.' 10		

Set up the system configuration

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

but the Watch Expression *(uchar)TEMPCOUNT* will not be effected by changing the value for the *Configure | Options | Expressions | Unknown data type size* command. The host software will always read just one byte and treat that byte as an **unsigned character** value.

What if the program is manipulating a 16-bit value at address 0x0050 in External Data memory (high byte at 0x0050, low byte at 0x0051) and there is no symbol associated with that location? Here we are about to add a Watch Expression that will let us monitor that value:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
				Source	File:demo.dbg		
Emulator			MOV	DPTR,#0000H	DPTR=0000h		
Mapping			CLR	T2	bit[90h]=1		
Code Memory			MOV	TEMPCOUNT,#0Ah	direct[64h]=00h		
Xdata Memory			LCALL	WASTETIME	10050		
Mode #1			CPL	T2			
Display			CLR	A			
Attributes			JNB	T2,SKIPOVER			
Windows			CPL	A			
Modify Ctrl-F6			DJNZ	TEMPCOUNT,INNERLOOP	10038 direct[64h]=0Ah		
Size Ctrl-F7			INC	DPTR			
Goto Ctrl-F8			LCALL	WASTETIME			
Repaint Alt-F1			LCALL	WASTETIME			
Watch			SJMP	OUTERLOOP			
			SJMP	START			
			MOV	R0,#0FFh	R0=64h		
				Add Watch Expression			
D:0064				(uint)x:50h	0A		
D:0064					0A '.' 10		
				Load			

Enter any expression denoting an address.

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

After the Watch Expression *(uint)x:50h* is added, the host software will monitor the two bytes starting at location 0x0050 in External Data memory and treat those two bytes combined as a 16-bit **unsigned integer** value. The default display format for such values is (unsigned) decimal notation, although you can change that to your choice of character, decimal or hexadecimal notations using the *Configure|Options|Expressions|int/uint* command:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
				Source	File:demo.dbg		
0030	900000	START:	MOV	DPTR,#0000H		DPTR=0000h	
0033	C290	OUTERLOOP:	CLR	T2		bit[90h]=1	
0035	75640A		MOV	TEMPCOUNT,#0Ah		direct[64h]=00h	
0038	120050	INNERLOOP:	LCALL	WASTETIME		10050	
003B	B290		CPL	T2			
003D	E4		CLR	A			
003E	309001		JNB	T2,SKIPOVER			
0041	F4		CPL	A			
0042	D564F3	SKIPOVER:	DJNZ	TEMPCOUNT,INNERLOOP	↑0038	direct[64h]=0Ah	
0045	A3		INC	DPTR			
0046	120050		LCALL	WASTETIME			
0049	120050		LCALL	WASTETIME			
004C	80E5		SJMP	OUTERLOOP			
004E	80E0	ENDOFPROGRAM:	SJMP	START			
0050	78FF	WASTETIME:	MOV	R0,#0FFh		R0=64h	
				Watch			
D:0064 (unknown)			TEMPCOUNT		0x0A		
D:0064 uchar			(uchar)TEMPCOUNT		0x0A '.'	10	
X:0050 uint			(uint)x:50h		65535		

Begin emulation

sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlobl SymAlph
1Help 2ResetEm3ResetTr4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr0StepTo

With the ability throughout the host software to specify addresses using expressions rather than just simple labels or absolute addresses, you can, for example, set break-points relative to a label:

Configure	File	Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
				Break-Points & Trace-On/Off			
				Add Remove Edit Numeric Symbolic Opcode_Class Load Save Window			
				ADD Start Address Skipover+0x20			
				End Address			
				(Hex or Symbolic)			
				Ctrl-R:Resize Ctrl-U:Move Tab,Shift-Tab:Select			
0041	F4		CPL	A		ACC=00h	
0042	D564F3	SKIPOVER:	DJNZ	TEMPCOUNT,INNERLOOP	↑0038	direct[64h]=09h	
0045	A3		INC	DPTR		DPTR=0000h	
0046	120050		LCALL	WASTETIME		10050	
0049	120050		LCALL	WASTETIME		10050	
004C	80E5		SJMP	OUTERLOOP			
004E	80E0	ENDOFPROGRAM:	SJMP	START			
0050	78FF	WASTETIME:	MOV	R0,#0FFh		R0=00h	
0052	D0FE		DJNZ	R0,0052h		←0052 R0=E9h	
0054	22		RET			ret addr=003Bh	
0055	FF		MOV	R7,A			
0056	FF		MOV	R7,A			
0057	FF		MOV	R7,A			
0058	FF		MOV	R7,A			

Edit address range

This is handy if you have a multi-module assembly language program and are using the assembler-generated listing files to set break-points. The listings contain addresses relative to the beginning of the module, not the final, absolute addresses in the bound (linked) program.

You can also set a break-point range based only on a "starting point" and a "distance", without having to compute the start and end points manually:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Break-Points & Trace-On/Off			
Add	Remove	Edit	Numeric Symbolic Opcode_Class Load Save Window
CBREAK	00	Start Address start End Address start+40h (Hex or Symbolic)	
Ctrl-R:Resize Ctrl-U:Move Tab,Shift-Tab:Select			
0041	F4	CPL A	ACC=00h
0042	D564F3	SKIPOVER: DJNZ TEMPCOUNT, INNERLOOP	10038 direct[64h]=09h
0045	A3	INC DPTR	DPTR=0000h
0046	120050	LCALL WASTETIME	10050
0049	120050	LCALL WASTETIME	10050
004C	80E5	SJMP OUTERLOOP	
004E	80E0	ENDOFPROGRAM: SJMP START	
0050	78FF	WASTETIME: MOV R0, #0FFh	R0=00h
0052	D8FE	DJNZ R0, 0052h	+0052 R0=E9h
0054	22	RET	ret addr=003Bh
0055	FF	MOV R7, A	
0056	FF	MOV R7, A	
0057	FF	MOV R7, A	
0058	FF	MOV R7, A	

Edit address range

You can specify ranges in the same way throughout the host software, no matter what you are doing:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help			
Source		File: demo.dbg	
0030	900000 START:	MOV DPTR, #0000H	DPTR=0000h
0033	C290 OUTERLOOP:	CLR T2	bit[90h]=1
0035	75640A	MOV TEMPCOUNT, #0Ah	direct[64h]=05h
0038	120050 INNERLOOP		10050
003B	B290		bit[90h]=1
003D	E4		ACC=FFh
003E	309001		bit[90h]=0
0041	F4		ACC=00h
0042	D564F3 SKIPOVER		t[64h]=09h
0045	A3		DPTR=0000h
0046	120050		10050
0049	120050		10050
004C	80E5		
004E	80E0 ENDOFPRO		R0=00h
0050	78FF WASTETIM		+0052 R0=E9h
0052	D8FE		ret addr=003Bh
0054	22		
0055	FF	MOV R7, A	
0056	FF	MOV R7, A	
0057	FF	MOV R7, A	
0058	FF	MOV R7, A	

View/Change Code Memory			
Browse	Fill	Move	Compare
Fill			
Address 0	From start	To start+0x500	Pattern 0
Esc:Cancel			
0000	02		
0008	FF		
0010	FF FF FF FF FF FF FF FF		
0018	FF FF FF FF FF FF FF FF		
0020	FF FF FF FF FF FF FF FF		
0028	FF FF FF FF FF FF FF FF		
0030	90 00 00 C2 90 75 64 0A		
0038	12 00 50 B2 90 E4 30 90		

Fill a block of Code memory

Chapter 3: Reference

Whenever and wherever the `INTEGER` host software requires an address, count or length, you can enter an arbitrary expression (suitable for the particular context of course). Expressions are composed of the following elements:

Identifiers

The first character in an identifier must be a digit or the `T` character.

- First character: `T @ A-Z`
- Subsequent characters: `T A-Z 0-9`

Constants

Integer Constants

`INTEGER` supports C-style constants as well as a variety of other popular formats. The following notations are used as subscripts: `A` means "one or more of the specified character or digit". A suffix of `-opt` means that the item is optional (i.e., "zero or more of").

- hex-digit: `0-9 A-F`
- dec-digit: `0-9`
- non-zero-dec-digit: `1-9`
- oct-digit: `0-7`
- bin-digit: `0-1`

Explicit Prefix The following constants contain an explicit radix (base) designator, either as a prefix or a suffix. Such constants will always be interpreted in the designated base, regardless of the current setting of the `Configure/Options/Prefixes/Basic` command (see page 3-12). Following each definition are some examples of that particular style of constant.

- hex-digit: `0x hex-digit`
- hex-digit: `0X hex-digit`

Hexadecimal constant with explicit prefix (C style).

An optional suffix of `W` or `U` specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

An optional suffix of `V` or `L` specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes `W`, `U`, `V` and `L` can be used at the same time, in either order.

- 0x12
- 0xa
- 0x123456

Expression Input

Whenever and wherever the *iceMASTER* host software requests an address, count or length, you can enter an arbitrary expression (suitable for the particular context, of course). Expressions are composed of the following elements.

Identifiers

The first character in an identifier name cannot be a digit or the '\$' character:

First character: _ ? @ a-z A-Z

Subsequent characters: _ \$ a-z A-Z 0-9

Constants

Integer Constants

iceMASTER supports C-style constants as well as a variety of other popular formats. The following notations are used in subsequent definitions. A plural suffix (*s*) means "one or more of" the specified characters or digits. A suffix of *-opt* means that the item is optional (i.e., "zero or more of").

hex-digit: 0-9 a-f A-F

dec-digit: 0-9

non-zero-dec-digit: 1-9

oct-digit: 0-7

bin-digit: 0-1

Explicit Radix The following constants contain an explicit radix (base) designator, either as a prefix or a suffix. Such constants will always be interpreted in the designated base, regardless of the current setting of the *Configure|Options|Expressions|Radix* command (see page 3-15). Following each definition are some examples of that particular style of constant.

0x *hex-digits*

0X *hex-digits*

Hexadecimal constant with explicit prefix (C style).

An optional suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

An optional suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes 'u'/'U' and 'l'/'L' can be used at the same time, in either order.

0x12

0xa

0x12751

X' hex-digits

Hexadecimal constant with explicit prefix (NSC style).

x'ab
x'45
x'123456

\$ hex-digits

Hexadecimal constant with explicit prefix (Motorola style).

\$ab
\$0045
\$123456

dec-digit hex-digits-opt h

dec-digit hex-digits-opt H

Hexadecimal constant with explicit suffix (Intel, et. al., style).

lah
0054h
12345678h

dec-digits t

dec-digits T

dec-digits .

Decimal constant with explicit suffix.

12t
045.

non-zero-dec-digit dec-digits-opt u

non-zero-dec-digit dec-digits-opt U

non-zero-dec-digit dec-digits-opt l

non-zero-dec-digit dec-digits-opt L

Decimal constant with explicit "suffix" (C style).

A suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

A suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes 'u'/'U' and 'l'/'L' can be used at the same time, in either order.

5u
23l
47ul

oct-digits o
oct-digits O
oct-digits q
oct-digits Q

Octal constant with explicit suffix.

12o
045o

0 *oct-digits*

Octal constant with explicit prefix (leading zero, C style).

An optional suffix of 'u' or 'U' specifies an unsigned integer. This is generally unnecessary, as the host software treats all constants as unsigned.

An optional suffix of 'l' or 'L' specifies a long (32-bit) integer rather than a 16-bit integer.

The suffixes 'u'/'U' and 'l'/'L' can be used at the same time, in either order.

012
055u

bin-digits y
bin-digits Y

Binary constant with explicit suffix.

1011y
011101Y

In all cases, if you do not supply an 'l' or 'L' suffix (or if such a suffix is not allowed) and the constant cannot fit in 16 bits, it will automatically be treated as a long value (32 bits).

If you are unsure of how a particular constant is being interpreted by the host software, just use the Display/Alter|Expression command (page 3-25) to see how the constant or constant expression is being treated.

No Explicit Radix

If you enter a number such as 10 or 101, it will be interpreted in the current default radix (base) as specified by the Configure|Options|Expressions|Radix command (see page 3-15). However, in certain contexts where it makes sense, the host software will temporarily override the current default. For example, when you enter a source line number as

#11

the software temporarily changes the default radix to 10 (decimal) before translating the '11'. In other cases, when the software displays a Dialog Box to obtain input, a prompt somewhere in the box (usually on the bottom border) indicates if the default radix has been changed temporarily.

In any case, you can always provide an explicit radix designator in the constant to override any default interpretation. For example, '#0x23' designates source line number 35 decimal, as does '#35t'.

In some cases, the software can automatically determine the appropriate radix based only on the characters and digits present in the constant. For example, 1A can only be the hexadecimal number 0x1A, even though no explicit prefix or suffix is present.

Floating-Point Constants

The software supports both single-precision and double-precision floating-point constants.

dec-digits . dec-digits exponent-opt
dec-digits exponent

The *exponent* has the following form:

exponent: *exp-char sign-opt dec-digits*

exp-char: **e** or **E**

sign: **+** or **-**

If there is no suffix, the constant is double-precision floating-point (**double**). An optional suffix of **'f'** or **'F'** specifies that the constant is single-precision floating-point (**float**). An optional suffix of **'l'** or **'L'** explicitly specifies explicitly that the constant is double-precision floating-point (**double**).

Unlike C, "1." is not a valid floating-point constant. You must always supply at least one digit after the decimal point (e.g., "1.0").

1.2
1.2f
5.6e5

Character Constants

The value of a character constant is the numerical value of the character interpreted as an integer. Internally, character constants are treated just as if you had entered an integer constant.

'\ 1-to-3-oct-digits '
'\x 1-to-3-hex-digits '
'\ escaped-char '
' char '

The first form uses an escaped sequence of one to three octal digits.

The second form uses an escaped sequence of one to three hexadecimal digits.

The third form specifies one of the following specially recognized escape characters:

Source Form	Internal Value	Meaning
<code>\a</code>	0x07	ANSI alert (bell)
<code>\b</code>	0x08	ANSI backspace
<code>\e</code>	0x1B	ANSI ESC
<code>\f</code>	0x0C	ANSI Formfeed
<code>\n</code>	0x0A	ANSI New line
<code>\r</code>	0x0D	ANSI Carriage return
<code>\t</code>	0x09	ANSI Horizontal tab
<code>\v</code>	0x0B	ANSI Vertical tab

If the backslash precedes a character that does not appear in the table above, the value used is the value of that character itself. That is, '\w' is the same as 'w'.

The fourth form represents the standard, "simple" case.

```
'\077'  
'\xAB'  
'\n'  
'H'
```

String Constants

"*chars-opt*"

The form of each *char* in *chars-opt* is the same as that allowed for character constants. That is, all forms of escape sequences are allowed internally.

Currently, string constants can only be used as fill values in the *Display/Alter|(memory)|Fill* commands.

Expressions

The operators and operands which are allowed in expressions are grouped below in decreasing precedence (priority) order. The highest precedence operators and expression elements are listed first. To simplify the description, the term *expr* means any expression at that precedence level or higher, and the term *expression* means any expression at all.

Primary identifier

Denotes a variable, label, module, function or SFR (Special Function Register) name. In the absence of any qualification on *identifier*, the software searches for the definition in the following order:

- local symbols in the current module
- global symbols

In the following descriptions, we use more meaningful names for identifiers to denote what kind of identifier is required in a particular context:

- *module-name*
- *function-name*
- *SFR-name*

constant

An integer constant, floating-point constant or character constant.

string

A character string constant (literal).

(*expression*)

A parenthesized expression. Parentheses can be used to change or clarify the order of operator precedence.

integer-constant

Source line number '*integer-constant*' in the current module.

module-name : # *integer-constant*

Source line number '*integer-constant*' in the module *module-name*.

module-name : *identifier*

Local variable or function *identifier* in module *module-name*.

module-name : *function-name* : *variable-name*

The local variable *variable-name* in the function *function-name* in module *module-name*.

C : *integer-constant*

X : *integer-constant*

D : *integer-constant*

I : *integer-constant*

B : *integer-constant*

Denotes a memory-space qualified constant. In most cases where the host software requests an address, you do not need to specify explicitly what memory space to use when you enter a constant or constant expression. For example, when setting a Code Memory break-point, if you enter **0x23**, the host software assumes that you mean location 0x0023 in Code Memory. Similarly, if you use labels or variable names, the symbol/debug information supplied in the program load module file normally tells in which memory space that symbol belongs.

However, if you want to change the value in a generic pointer (see page 3-13), you must tell the host software the memory space in which the address exists (which memory space the address is relative to).

Designator	Memory Space	Applicability			
		MCS-51	COP8	68HC11	68HC05
C	Code	x	x	x	x
D	Directly-addressable Internal Data	x	x		
I	Indirectly-addressable Internal Data	x			
X	External Data	x			
B	Bit-addressable Internal Data	x			

You can use a memory-space qualified constant to designate an address in other contexts, too. For example, in a program with no symbolic debug information, you can add the Watch Expression **X:0x23** to the Watch Window. This will allow you to monitor location 0x0023 in External Data memory.

X:0x23 designates location 0x0023 in External Data memory

D:45h designates location 0x45 in Directly-addressable Internal Data/RAM memory

expr . *member-name*

expr . integer-constant

(MCS-51 only) Designates a particular bit in a bit-addressable SFR. *integer-constant* must be between 0 and 7 inclusive.

The *expr* must have the type "pointer to **struct**" or "pointer to **union**".

have type "array of ..." or "pointer to
l type.

* *expr*

Indirection through pointer expression *expr*, which must be of a "pointer to ..." type.

- *expr*

Unary minus. The *expr* can be either integral or a floating-point constant.

$$+ \text{expr}$$

(*type-name*) *expr*

(*type-name*) *expr*

Casting is intended for use in conjunction with a memory-space qualified constant (an address). The meaning of such an expression is "treat the location *expr* as having data type *type-name*".

The available *type-names*, which are reserved words, are:

<i>type-name</i>	C Data Type	Size	Memory Layout
bit	bit	1 bit	
char	char	1 byte	
uchar	unsigned char	1 byte	
short	short	2 bytes	MSB, LSB
ushort	unsigned short	2 bytes	MSB, LSB
int	int	2 bytes	MSB, LSB
uint	unsigned int	2 bytes	MSB, LSB
long	long	4 bytes	MSW, LSW (each MSB, LSB)
ulong	unsigned long	4 bytes	MSW, LSW (each MSB, LSB)
float	float	4 bytes	IEEE single-precision floating-point
double	double	8 bytes	IEEE double-precision floating-point

MSB: Most Significant Byte (8 bits)

LSB: Least Significant Byte (8 bits)

MSW: Most Significant Word (16 bits, 2 bytes)

LSW: Least Significant Word (16 bits, 2 bytes)

Note that no conversions are performed on the value itself. The cast merely changes the data type that the host software associates with the designated address expression.

Multiplicative *expr * expr*

Multiplication.

expr / expr

Division.

expr % expr

Modulo division.

Additive

expr + expr

Addition.

expr - expr

Subtraction.

Display Formats

This section describes the display formatting conventions used by the host software.

Addresses

Addresses displayed in the following contexts

- Watch Window
- *Display/Alter|Expression* command
- *Source/Symbols|Address* command
- *Source/Symbols|Alpha* command
- *Source/Symbols|Global* command
- *Source/Symbols|Local* command
- Pop-up Change Box
- Pop-up Browse Window

are of the form:

m:hhhh

where *m* is a character denoting the memory space and *hhhh* is the hexadecimal offset into that memory space. The possible values of *m* are:

Designator <i>m</i>	Memory Space	Applicability			
		MCS-51	COP8	68HC11	68HC05
C	Code	x	x	x	x
D	Directly-addressable Internal Data	x	x		
I	Indirectly-addressable Internal Data	x			
X	External Data	x			
B	Bit-addressable Internal Data	x			
N	(none / any / "number")	x	x	x	x

Type Names

Fundamental Types

Displayed Type Name	C Data Type	Size (Memory Layout)
(unknown)	-	<i>Configure Options Expressions Unknown data type size</i> (see page 3-16)
(untyped)	-	
bit	bit	1 bit
char	signed char	1 byte
uchar	unsigned char	1 byte
short	signed short	2 bytes (MSB, LSB)
ushort	unsigned short	2 bytes (MSB, LSB)
int	signed int	2 bytes (MSB, LSB)
uint	unsigned int	2 bytes (MSB, LSB)
long	signed long	4 bytes (MSW (MSB, LSB), LSW (MSB, LSB))
ulong	unsigned long	4 bytes (MSW (MSB, LSB), LSW (MSB, LSB))
float	float	4 bytes (IEEE format)
double	double	8 bytes (IEEE format)
void	void	-
(label)	(statement label)	-

Bit Field Types

The type name displayed for structure members which are C bit fields shows the underlying integral type preceded by the bit offset into an object of that type and the number of bits in the field. The general format is:

`offset:length_type-name`

where 'offset' is the offset to the least significant bit in the field, 'length' is the field length (in bits) and *type-name* is the name of the underlying type. The offset 'offset' is zero-relative, starting at the least significant bit. For example, the bits in a **char** are numbered 7,...,0, with 7 being the most significant bit and 0 being the least significant bit.

See page 3-18 for an example of the display of a structure containing bit field members.

Derived/Aggregate Types

The general display formats are as follows, where "..." represents any data type name:

Display Format	Meaning
-> ...	"pointer to ..."
[n]...	"array of n ..."
()...	"function returning ..."
struct <i>tag-name</i>	(structure with tag name <i>tag-name</i>)
struct	(structure with no tag name)
union <i>tag-name</i>	(union with tag name <i>tag-name</i>)
union	(union with no tag name)

Unlike C, where data types read "from the inside out, obeying the parentheses", the data type names displayed by the host software read strictly from left-to-right:

Displayed Type Name	Example C Declaration	Data Type
-> int	int* pi;	"pointer to int "
[5]int	int x[5]	"array of 5 ints "
()int	int f();	"function returning int "
[15]-> int	int* y[15];	"array of 15 pointers to int "
-> ()-> [10]-> char	char* (*(abc())[10];	"pointer to a function returning a pointer to an array of 10 pointers to char "

Values

The display format of a value depends on that value's data type.

Fundamental Types

You can display values of the fundamental types in one of three formats:

- Character (enclosed in apostrophes ('))
- Decimal (signed or unsigned)
- Hexadecimal

You control the display format for a particular data type with commands in the *Configure|Options|Expressions* pull-down menu (see page 3-20).

Pointers

The display format of a pointer value depends on the "nature" of the pointer.

Generic Pointers

Generic pointers are needed due to the Harvard architecture of the MCS-51. A generic pointer occupies three bytes in memory. The first byte contains a selector value/code denoting which memory space is being referenced (e.g., Code memory, External Data memory, etc.). The next two bytes contain the offset (0x0000-0xFFFF) into that memory space.

A generic pointer is used when the compiler does not know (at compile-time) what memory space a pointer variable may reference.

The general display format for a generic pointer is:

m:hhhh

where *m* is a character denoting the memory space and *hhhh* is the hexadecimal offset into that memory space. The possible memory space characters (*m*) are a subset of those listed in the table on page 3-10.

The display format for invalid (illegal, erroneous) generic pointers is:

?:hhhh {sshhhh}

An invalid generic pointer value is one with an illegal memory space selector byte value. This can occur for several reasons:

- 1) The application program has not yet initialized (assigned to) the pointer.
- 2) The application program has erroneously "clobbered" (written to) the memory space selector byte value in the pointer.
- 3) The linker has assigned (static) variables to memory locations so that local variables in functions which cannot be active at the same time share the same memory for their local variables. For example, if functions 'a' and 'b' can never be active at the same time, the linker may assign the local variables in function 'a' such that they use the same memory locations as the local variables in function 'b'. If function 'b' is currently active and you display a local pointer variable defined in function 'a', the value in that pointer variable may appear to be "bad".

Memory-Specific Pointers

Memory-specific pointers are used when the compiler "knows" at compile-time what memory space is being referenced by a particular pointer. Memory-specific pointers occupy only the number of bytes required to hold an address spanning a particular memory space. Memory-specific pointers do not contain a memory space selector byte.

The general display formats for memory-specific pointers are:

(m):hh

(m):hhhh

where *m*, *hh* and *hhhh* are the same as for the generic pointers. The memory space designator *m* is included to let you know which memory space is being referenced by the offset value *hh* or *hhhh*. It is enclosed in parentheses to let you know that the memory space is implicit (i.e., there is no memory space selector byte stored in memory).

NULL Pointers

The display format for a NULL pointer value is "(null)". The rules for detecting a NULL pointer value depend on the compiler you are using. See page 3-29 for specific details about NULL pointer values.

Structures, Unions and Arrays

The single-line "value" displayed for a structure, union or array object is the data type name itself ("struct...", "union..." or "[n]...").

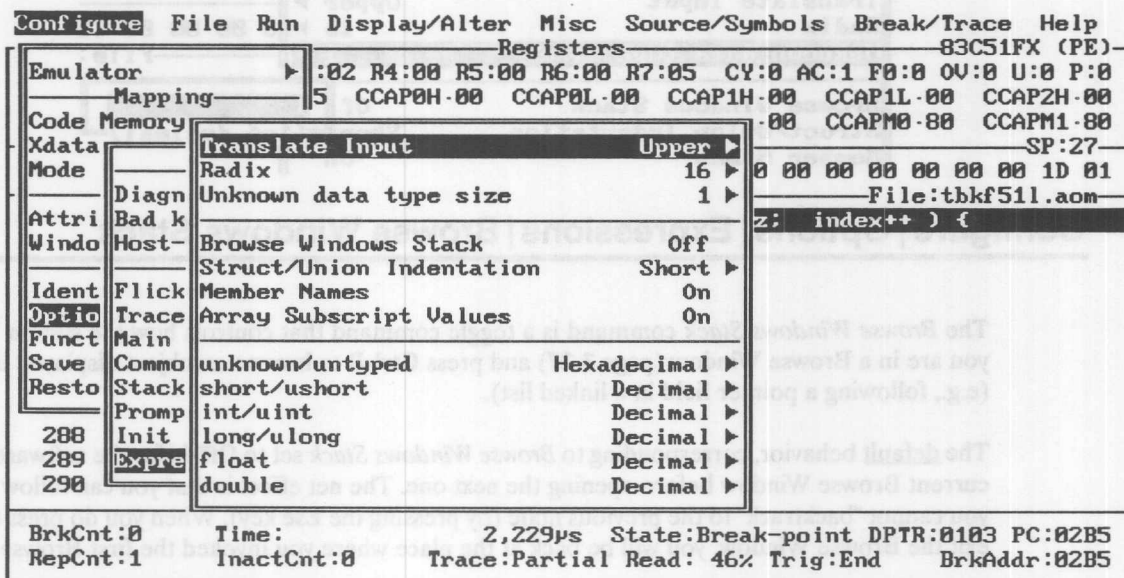
In the main-screen Watch Window and the pop-up Browse Windows, structures, unions and arrays are fully-expanded to show the individual members or elements. In these contexts, the individual members in a **struct/union** and the individual elements in an array immediately follow the single-line header for the structure, union or array. Each of these lines show the address, data type and value of the **struct/union** member or array element.

Functions

The value displayed for an entity of type "function returning..." is the name of the function itself. This may seem redundant and silly in the simple case, but if you are following (indirecting through) a pointer to a function, the software displays the name of the function pointed to by the pointer.

Configure | Options | Expressions Command

The *Configure | Options | Expressions* command displays a pull-down menu of commands to allow you to control expression input (symbols and numbers), numeric value formatting, data structure formatting and data structure browsing features.

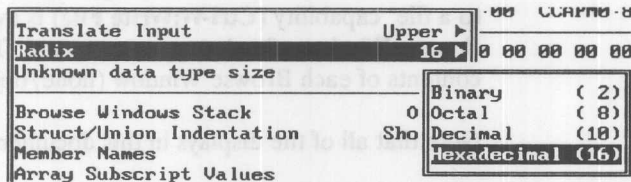


Case conversion to be applied to keyboard input and symbolic input from files

Configure | Options | Expressions | Radix

The *Radix* command allows you to specify the default radix (base) for numbers you enter that contain no explicit radix prefix or suffix. If you enter a number such as 10 or 101, it will be interpreted in the current default radix (base) as specified by this command. However, in certain contexts where it makes sense, the host software will temporarily override the current default. For example, when you enter a source line number as

#11



the software temporarily changes the default radix to 10 (decimal) before translating the '11'. In other cases, when the software displays a Dialog Box to obtain input, a prompt somewhere in the box (usually on the bottom border) indicates if the default radix has been changed temporarily.

In any case, you can always provide an explicit radix designator in the constant to override any default interpretation. For example, '#0x23' designates source line number 35 decimal, as does '#35t'.

In some cases, the software can automatically determine the appropriate radix based only on the characters and digits present in the constant. For example, 1A can only be the hexadecimal number 0x1A, even though no explicit prefix or suffix is present.

Configure | Options | Expressions | Unknown data type size

The *Unknown data type size* command tells the host software how many bytes to read, write or display for a value/symbol whose data type is "(unknown)" or "(untyped)".

Translate Input	Upper ▶	00 CCAPM0·8
Radix	16 ▶	0 00 00 00 00
Unknown data type size	1 ▶	File:
Browse Windows Stack	Of	1
Struct/Union Indentation	Shor	(1-4 decimal)
Member Names	On	

Configure | Options | Expressions | Browse Windows Stack

The *Browse Windows Stack* command is a toggle command that controls how the system behaves when you are in a Browse Window (page 3-27) and press **Ctrl-B** to browse an object displayed in that window (e.g., following a pointer field in a linked list).

The default behavior, corresponding to *Browse Windows Stack* set to **Off**, is for the software to "close" the current Browse Window before opening the next one. The net effect is that you can follow a pointer, but you cannot "backtrack" to the previous node (by pressing the Esc key). When you do press the Esc key to exit the Browse Window, you will be back at the place where you invoked the first Browse Window.

A *Browse Windows Stack* setting of **On** causes successive pop-ups of a Browse Window to stack one on top of the other (recursively). You can backtrack to any of the previous Browse Windows by pressing the **Esc** key one or more times. As nice as it is, the reason that this is not the default system behavior is that you could cause the host software to overflow its own runtime stack by traversing too deeply into list or tree or whatever.

However, even when you have *Browse Windows Stack* set to **Off** (default setting), there is a way to at least maintain a record of the nodes/objects you have visited while browsing. The "write the data in this window to a file" capability (**Ctrl-W:Write File**) is available in a Browse Window, regardless of the setting of the *Browse Windows Stack* toggle command. Thus, at each step along the way, you can write the entire contents of each Browse Window (node/object), successively appending to the same file.

Note that all of the displays in this document were generated with *Browse Windows Stack* set to **On**.

Configure | Options | Expressions | Struct/Union Indentation

The *Struct/Union Indentation* command controls the indentation of structure and union members which are themselves structures, unions or arrays. The two indentation options available are 'Short' and 'Long'. The default is 'Short' indentation.

With 'Short' indentation, the members/elements of embedded structures, unions or arrays are indented two spaces from the beginning of the name of the "parent" member:

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	lf_link	union
X:0104	->struct Srt_N	srx_left	X:0124
X:0104	->struct Srt_N	srx_fore	X:0124
X:0107	union	rb_link	union
X:0107	->struct Srt_N	srx_right	X:0144
X:0107	->struct Srt_N	srx_back	X:0144
X:010A	[4]char	srt_value	[4]char
X:010A	char	[0] 0x41	'A' 65
X:010B	char	[1] 0x42	'B' 66
X:010C	char	[2] 0x43	'C' 67
X:010D	char	[3] 0x00	'.' 0
X:010E	uchar	srt_vtype	0x04 '.' 4
X:010F	char	srt_char	0x4D 'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

With 'Long' indentation, the members/elements of embedded structures, unions or arrays are indented two spaces from the beginning of the value field of the "parent" member:

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	lf_link	union
X:0104	->struct Srt_N	srx_left	X:0124
X:0104	->struct Srt_N	srx_fore	X:0124
X:0107	union	rb_link	union
X:0107	->struct Srt_N	srx_right	X:0144
X:0107	->struct Srt_N	srx_back	X:0144
X:010A	[4]char	srt_value	[4]char
X:010A	char	[0] 0x41	'A' 65
X:010B	char	[1] 0x42	'B' 66
X:010C	char	[2] 0x43	'C' 67
X:010D	char	[3] 0x00	'.' 0
X:010E	uchar	srt_vtype	0x04 '.' 4
X:010F	char	srt_char	0x4D 'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

Note that this command has a visible effect only if the setting of the *Configure | Options | Expressions | Member Names* toggle command is 'On' (see page 3-18).

Configure | Options | Expressions | Member Names

The *Member Names* toggle command controls whether or not the host software will display structure and union member names when displaying the individual members of a structure or union. The default is 'On'.

This is an example of a union display generated with the *Member Names* toggle set to 'On' (the default):

Display/Alter Expression				
Expression: b_flags1_1				
Addr: X:00FF Size: 1 byte Type: union B_Flags1				
X:00FF union B_Flags1 b_flags1_1				
	Addr	Data Type	Value	
	X:00FF	union B_Flags1	union B_Flags1	
	X:00FF	uchar	full_val 0x73 's'	115
K_F	X:00FF	struct	b	struct
ags1_2.b	X:00FF	0:1_uchar	bf_1 0x01 '.'	1
	X:00FF	1:1_uchar	bf_2 0x01 '.'	1
Time:	X:00FF	2:1_uchar	bf_3 0x00 '.'	0
InactCnt	X:00FF	3:3_uchar	bf_4 0x06 '.'	6

Here is what the same display looks like with the *Member Names* toggle set to 'Off':

Display/Alter Expression			
Expression: b_flags1_1			
Addr: X:00FF		Size: 1 byte	Type: union B
X:00FF union B_Flags1 b_flags1_1			
Addr	Data Type	Value	
X:00FF	union B_Flags1	union B_Flags1	
X:00FF	uchar	0x73 's'	115
K_F	struct	struct	
ags1_2.b	X:00FF 0:1_uchar	0x01 '.'	1
	X:00FF 1:1_uchar	0x01 '.'	1
Time:	X:00FF 2:1_uchar	0x00 '.'	0
InactCnt	X:00FF 3:3_uchar	0x06 '.'	6

Configure | Options | Expressions | Array Subscript Values

The *Array Subscript Values* toggle command controls whether or not the host software will display an array element's subscript value (enclosed in square brackets) before the actual value of the array element itself. The default is 'On'.

This is an example of an array display generated with the *Array Subscript Values* toggle set to 'On' (the default):

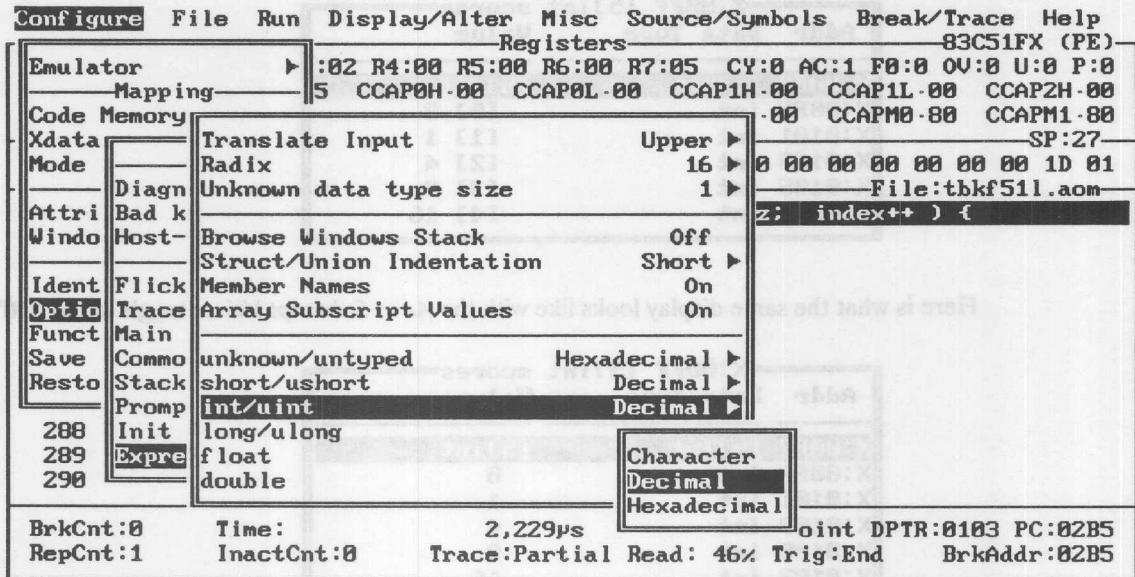
X:00FF [5]int scores		
Addr	Data Type	Value
X:00FF	[5]int	[5]int
X:00FF	int	[0] 0
X:0101	int	[1] 1
X:0103	int	[2] 4
X:0105	int	[3] 9
X:0107	int	[4] 16

Here is what the same display looks like with the *Array Subscript Values* toggle set to 'Off':

X:00FF [5]int scores		
Addr	Data Type	Value
X:00FF	[5]int	[5]int
X:00FF	int	0
X:0101	int	1
X:0103	int	4
X:0105	int	9
X:0107	int	16

Configure	Options	Expressions	unknown/untyped
Configure	Options	Expressions	short/ushort
Configure	Options	Expressions	int/uint
Configure	Options	Expressions	long/ulong
Configure	Options	Expressions	float
Configure	Options	Expressions	double

These commands control the display format to be used for values of the fundamental (basic) data types.



Display in decimal format

The available choices are:

Command	Controls Display Formatting for These Data Types	Display Formatting Options Available
<i>Configure Options Expressions unknown/untyped</i>	(unknown) & (untyped)	Character, Decimal (signed or unsigned), or Hexadecimal
<i>Configure Options Expressions short/ushort</i>	short & unsigned short	
<i>Configure Options Expressions int/uint</i>	int & unsigned int	
<i>Configure Options Expressions long/ulong</i>	long & unsigned long	
<i>Configure Options Expressions float</i>	float	
<i>Configure Options Expressions double</i>	double	

Note that there is no control needed for the **char** and **unsigned char** types, since values of these 1-byte types are always displayed in all three possible formats: hexadecimal, character and signed/unsigned decimal (in that order).

If you select Decimal format, signed or unsigned decimal notation will be used depending on the actual data type of the value (e.g., signed decimal for **int**, unsigned decimal for **unsigned int**).

Configure | Options | Expressions Command Combinations

The *Configure* | *Options* | *Expressions* commands operate, for the most part, independently of one another.

For example, the following structure display was generated with everything set to the "maximum" (*Struct/Union Indentation* = **Long**, *Member Names* = **On**, *Array Subscript Values* = **On**):

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	union	
X:0104	->struct Srt_N	srx_left X:0124	
X:0104	->struct Srt_N	srx_fore X:0124	
X:0107	union	union	
X:0107	->struct Srt_N	srx_right X:0144	
X:0107	->struct Srt_N	srx_back X:0144	
X:010A	[4]char	[4]char	
X:010A	char	[0] 0x41	'A' 65
X:010B	char	[1] 0x42	'B' 66
X:010C	char	[2] 0x43	'C' 67
X:010D	char	[3] 0x00	'.' 0
X:010E	uchar	srt_vtype 0x04	'.' 4
X:010F	char	srt_char 0x4D	'M' 77
X:0110	int	srt_int	25
X:0112	uint	srt_flags	20
X:0114	long	srt_long	25
X:0118	ulong	srt_ulong	25
X:011C	float	srt_float	12345.678

Here is the same structure display generated with everything set to the "minimum" (*Struct/Union Indentation* = **Short**, *Member Names* = **Off**, *Array Subscript Values* = **Off**):

X:0164 ->struct Srt_Nd newnode			
Addr	Data Type	Value	
X:0104	struct Srt_Nd	struct Srt_Nd	
X:0104	union	union	
X:0104	->struct Srt_N	X:0124	
X:0104	->struct Srt_N	X:0124	
X:0107	union	union	
X:0107	->struct Srt_N	X:0144	
X:0107	->struct Srt_N	X:0144	
X:010A	[4]char	[4]char	
X:010A	char	0x41	'A' 65
X:010B	char	0x42	'B' 66
X:010C	char	0x43	'C' 67
X:010D	char	0x00	'.' 0
X:010E	uchar	0x04	'.' 4
X:010F	char	0x4D	'M' 77
X:0110	int	25	
X:0112	uint	20	
X:0114	long	25	
X:0118	ulong	25	
X:011C	float	12345.678	

Watch Window

Commands The *Configure|Windows|Watch* pull-down menu contains five commands to allow you to define and manipulate the content of the Watch Window.

The **Add** command lets you add any non-constant expression to the Watch Window. Note that the host software reevaluates the address designated by the watch expression each time the Watch Window is updated. Thus, if you add the expression *my_array[i]* to the Watch Window, the host software recomputes the subscript value of *i* each time the software updates or repaints the Watch Window.

The **Delete** command deletes (removes) a single, designated expression from the Watch Window.

The **Clear** command deletes (removes) all watch expressions from the Watch Window.

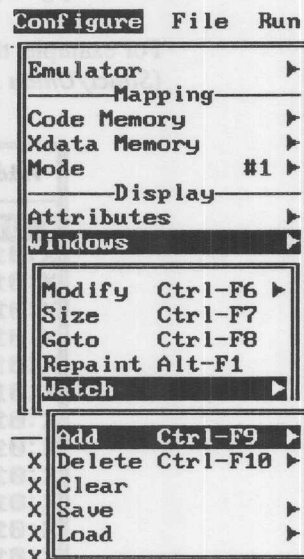
The **Save** command saves (writes) all the watch expressions in the Watch Window to a file of your choice. You can later load (add) these saved watch expressions to the Watch Window by selecting the *Configure|Windows|Watch|Load* command below.

The **Load** command loads (restores) into the Watch Window the watch expressions saved in a file by the *Configure|Windows|Watch|Save* command above.

The various commands, or the entire menu, may be inactive (inaccessible) depending on the current state of the system. For example, if the Watch Window is empty, the Delete, Clear and Save commands will be inactive (inaccessible).

Display Format

The Watch Window has four columns, showing the address of the watch expression, the data type of the watch expression, the watch expression itself and the current value of the watch expression:



Configure File				Run	Display/Alter	Misc	Source/Symbols	Break/Trace	Help
Module:TA		Source		File:takf51l.aom					
76	aj = (ai * 2) + main_loop_factor;								
77	ary[ail = aj;								
Watch									
X:008E	uint	ai	3						
X:008A	uchar	ary[ail	0x2F '/' 47						
X:0084	[10]uchar	ary	[10]uchar						
X:0084	uchar	[0]	0x24	'\$'	36			
X:0085	uchar	[1]	0x26	'&'	38			
X:0086	uchar	[2]	0x28	'('	40			
X:0087	uchar	[3]	0x2A	'*'	42			
X:0088	uchar	[4]	0x2C	','	44			
X:0089	uchar	[5]	0x2E	'.'	46			
X:008A	uchar	[6]	0x2F	'/'	47			
X:008B	uchar	[7]	0x31	'1'	49			
X:008C	uchar	[8]	0x33	'3'	51			
X:008D	uchar	[9]	0x35	'5'	53			
X:00A8	->struct List	p_list	(null)						
C:0000?	struct List	*p_list	struct List						
C:0000?	uint	list_uival 517						
C:0002?	->struct List	list_p_nxt ? :FFFF{CEFFFF}						
X:0080	float	iteration	38.5						

Begin emulation

If the watch expression is a structure, union or array type, then there will also be a line for each constituent component in the watch expression: each member in a **struct/union** or each element in an array. In this case, the watch expression field contains ".....".

Every time the host software updates or repaints the Watch Window, it highlights the value field of each watch expression or constituent component if the current value is different from the previous value. This highlighting is done to notify you of the change in value. Similarly, if the address associated with the watch expression or constituent component changes, the software highlights the address field. (The address associated with a watch expression can change if you are watching an expression such as *my_array[i]*, where the actual array element being monitored changes as the value of *i* changes.)

If the host software encounters an "illegal" value in evaluating the address of a watch expression, it flags the watched object with a question mark (?) between the address column and the data type column. This can happen if the watch expression involves indirection through a pointer and the value of that pointer is NULL or illegal (see page 3-13).

Browsing

You can enter Change Mode to browse the Watch Window by pressing the **Tab** key (which is simply a convenient shortcut for the *Configure|Windows|Goto* command) one or more times until the Watch Window title is highlighted:

```

Module:TA                                     Source                                     File:takf511.aom
90      /* scan through nodes in list */
91      p_list = list_head;
92      while (p_list != NULL) {
93          lj = p_list->list_uival;
94          p_list = p_list->list_p_nxt;
95      } /*while p_list*/
96
97      iteration += 1.1;
98      main
99      Addr  Data Type  Value
100 End_Loop: ;
101      X:0097 struct List  struct List
102      } /* X:0097 uint  list_uival 142
103      X:0099 ->struct List  list_p_nxt X:0092
104 } /*main*/
Enter:Change
X:00A8 ->struct List  p_list  X:009C
X:009C struct List  *p_list  struct List
X:009C uint  list_uival 146
X:009E ->struct List  list_p_nxt X:0097
X:0080 float  iteration  150.7
Ctrl-B:Browse

```

Tab/Shift-Tab:Next/Prev window (Keypad):Scroll Esc:Exit

As you scroll up or down in the Watch Window, the software will display prompts on the top border of the Watch Window indicating what actions you can perform on the highlighted item:

Enter:Change - (upper-left border) The currently highlighted item is a fundamental value or a pointer, either of which you can change. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the item's current value.

Ctrl-B:Browse - (upper-right border) The currently highlighted item is a pointer which can be "followed". Pressing the **Ctrl-B** key sequence pops up a Browse Window (see page 3-27), allowing you to browse/inspect the object pointed to by the pointer value.

Source/Symbols Displays

The following *Source/Symbols* windows display the internal symbol table sorted in various ways:

Source/Symbols|Global: global (PUBLIC) symbols, sorted alphabetically

Source/Symbols|Local: local symbols, sorted alphabetically within each scope

Source/Symbols|Alpha: global and local symbols, sorted alphabetically

Source/Symbols|Address: global and local symbols, sorted by address

All these windows have the same basic display format. Following is a display generated by the *Source/Symbols|Local* command:

Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help				
Module:TB		Source		File:tbkf511.aom
Local Symbols - By Module				
Enter Symbol		(104 Symbols)		
Addr	Symbol Name (M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
C:04DD	arrays	()void	arrays	1 TB
	F MAIN			
X:00A1	ages	[20]int	[20]int	2 MAIN
X:00C9	average	int	-1233	2 MAIN
X:00ED	dsp_ftn	->()float	C:033A	2 MAIN
X:00CF	per_1	struct per	struct per	2 MAIN
X:00EA	per_p1	->struct per	X:00CF	2 MAIN
X:00F0	sptr	->struct Srt_N	(null)	2 MAIN
X:00F3	textbook	struct book	struct book	2 MAIN
X:009D	x	int	5	2 MAIN
X:009F	y	int	10	2 MAIN
X:00CB	z	float	12345.678	2 MAIN
	F DO_NOTHING			
B:0028	key	bit		1 2 DO_NOTHING
Enter:Change Ctrl-R:Resize Ctrl-U:Move Ctrl-B:Browse				

Source level debug and symbol displays

The information presented for each symbol consists of memory address, symbol name, data type, current value and the scope in which the symbol was defined. Each symbol occupies one line in these displays. If the symbol is a structure, union or array, its constituent components (**struct/union** members or array elements) are not shown in these displays. To see the individual **struct/union** members or array elements, you must press the "browse" key combination (**Ctrl-B:Browse** - see below).

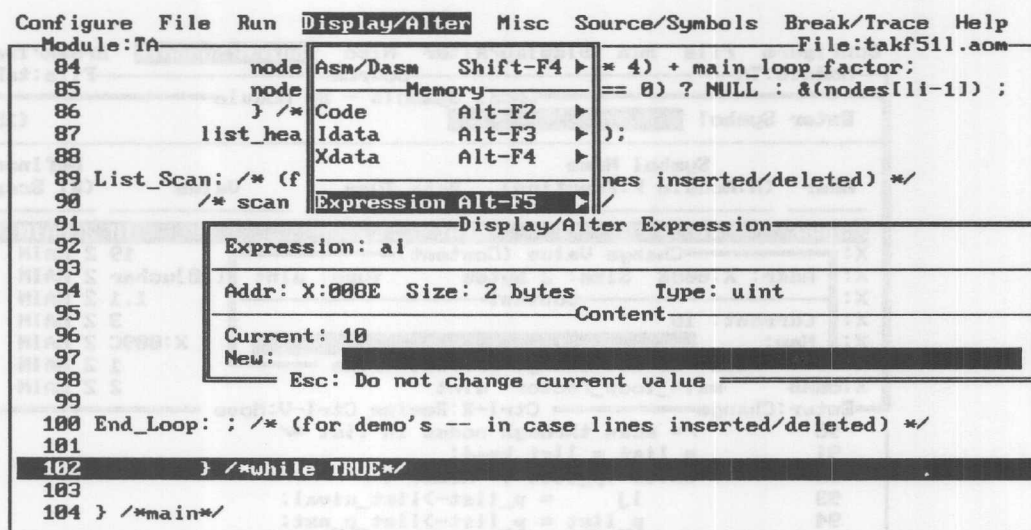
As you scroll up or down in these *Source/Symbols* windows, the software will display prompts on the bottom border of the window indicating what actions you can perform on the highlighted symbol:

Enter:Change - (bottom-left border) The currently highlighted symbol's data type is one of the fundamental types or it is a pointer. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the symbol's current value.

Ctrl-B:Browse - (bottom-right border) The currently highlighted symbol is an array, structure or union, or it is a pointer (which can be "followed"). Pressing the **Ctrl-B** key sequence pops up a Browse Window (see page 3-27), allowing you to browse/inspect the array elements, **struct/union** members or the object pointed to by the pointer variable.

Display/Alter | Expression

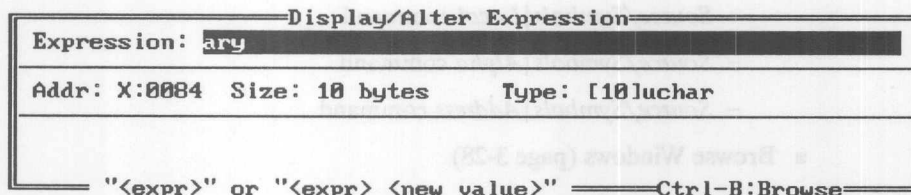
The *Display/Alter|Expression* command pops up a dialog box allowing you to view or change the value stored in a location designated by an arbitrary expression.



View/change the value in any variable, register, bit, memory location, etc.

The dialog box shows the address, size (in bytes or bits) and data type of the expression you enter. If the expression designates a location whose content can be changed, the "Current:" field shows the value currently stored in that location.

If the expression you enter designates a structure, union or array value, the software shows the address, size and data type of the expression. Additionally, the **Ctrl-B:Browse** prompt appears on the bottom-right border of the box to indicate that you can pop up a Browse Window (page 3-27) to inspect and/or change the individual members in the **struct/union** or the individual elements in the array:



If the expression you enter designates a "pointer to..." value, you have the option of either changing the pointer, by entering a new value, or "following" the pointer, by pressing **Ctrl-B** (Browse).

Note: Currently, the expression entered in the "New:" field is evaluated for address, not for value. For example, if you enter *i*, the address of the variable *i* is used as the new value, not the content of the variable *i*.

Change Box

A Change Box is very much like the *Display/Alter|Expression* command's pop-up dialog box. The difference is that, with a Change Box, the expression or variable whose value is to be changed is already known.

The screenshot shows the Source/Symbols window with the following menu bar: Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help. The window title is "Module:TA" and the file is "File:takf511.aom". The window is divided into two panes. The top pane is titled "Local Symbols - By Module" and contains a table of symbols. The bottom pane shows the source code for the module.

Addr	Symbol Name (M:Module F:Function)	Data Type	Value	Defined Within (#: Scope Level)
X:008E	ai	uint	10	2 MAIN
X:	Change Value (Content)		19	2 MAIN
X:	Addr: X:008E Size: 2 bytes Type: uint		[10]uchar	2 MAIN
X:	Content		1.1	2 MAIN
X:	Current: 10		3	2 MAIN
X:	New:		X:009C	2 MAIN
X:	Esc: Do not change current value		1	2 MAIN
X:00AB	main_loop_factor	uint	2	2 MAIN

Enter Symbol (15 Symbols)

Enter:Change Ctrl-R:Resize Ctrl-U:Move

```

90 /* scan through nodes in list */
91 p_list = list_head;
92 while (p_list != NULL) {
93     lj = p_list->list_uival;
94     p_list = p_list->list_p_nxt;
95 } /*while p_list*/

```

Source level debug and symbol displays

A Change Box pops up when you press the **Enter** key in response to the **Enter:Change** prompt in the following contexts:

- Watch Window (page 3-23)
- Source/Symbols windows (page 3-24)
 - Source/Symbols|Global command
 - Source/Symbols|Local command
 - Source/Symbols|Alpha command
 - Source/Symbols|Address command
- Browse Windows (page 3-28)

Browse Windows

A Browse Window pops up to allow you to view (inspect) and/or change the

- members in a structure or union
- elements in an array
- object designated by a pointer value (i.e., to "follow" the pointer)

Here is a Browse Window popped-up from the *Display/Alter|Expression* command to examine the individual elements in the array *ary*:

The screenshot shows a debugger window titled "Display/Alter" with a menu bar: Configure, File, Run, Display/Alter, Misc, Source/Symbols, Break/Trace, Help. The main area is divided into two panes. The left pane shows a C code snippet with line numbers 75 to 95. Line 82, "Expression: ary", is selected. The right pane shows a table of memory data for the array 'ary' at address X:0084. The table has columns for Address, Data, Type, and Value. The data is as follows:

Addr	Data	Type	Value
X:0084	[10]uchar		[10]uchar
X:0084	uchar		[0] 0x02 , , 2
X:0085	uchar		[1] 0x04 , , 4
X:0086	uchar		[2] 0x06 , , 6
X:0087	uchar		[3] 0x08 , , 8
X:0088	uchar		[4] 0x0A , , 10
X:0089	uchar		[5] 0x0C , , 12
X:008A	uchar		[6] 0x0E , , 14
X:008B	uchar		[7] 0x10 , , 16
X:008C	uchar		[8] 0x12 , , 18
X:008D	uchar		[9] 0x14 , , 20

Display/Alter memory

A Browse Window pops up when you press the **Ctrl-B** key in response to the **Ctrl-B:Browse** prompt in the following contexts:

- Watch Window (page 3-23)
- Source/Symbols windows (page 3-24)
 - *Source/Symbols|Global* command
 - *Source/Symbols|Local* command
 - *Source/Symbols|Alpha* command
 - *Source/Symbols|Address* command
- *Display/Alter|Expression* command (page 3-25)
- Browse Windows (e.g., to follow a pointer which is a **struct/union** member or an array element)

Structures, unions and arrays are fully "expanded" in a Browse Window to show all constituent components: each member in a **struct/union** or each element in an array. This process applies iteratively if a component is itself a structure, union or array.

The *Configure|Options|Expressions|Browse Window Stack* toggle command (page 3-16) controls how the system behaves when you are in a Browse Window and press **Ctrl-B** to browse an object displayed in that window (e.g., following a pointer field in a linked list).

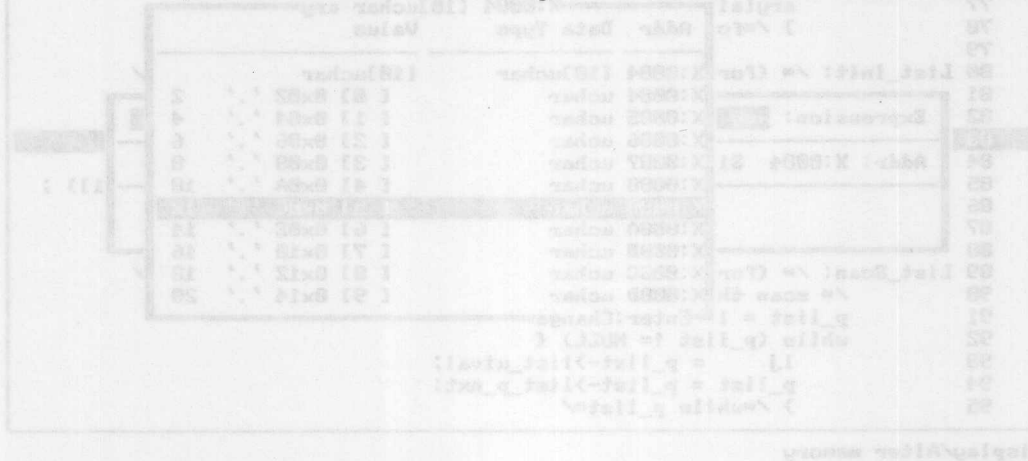
The following keys are available in a Browse Window:

Enter:Change – (lower-left border) The currently highlighted item is a fundamental value or a pointer, either of which you can change. Pressing the **Enter** key pops up a Change Box (see page 3-26), allowing you to change the item's current value.

Ctrl-B:Browse – (lower-right border) The currently highlighted item is a pointer which can be "followed". Pressing the **Ctrl-B** key sequence pops up another Browse Window, allowing you to browse/inspect the object pointed to by the pointer value.

Ctrl-W:Write – (upper-right border) Pressing **Ctrl-W** allows you to write the entire contents of the Browse Window to a file of your choice. The software will prompt you for the name of the file and ask you whether you want to append to that file (add to the end of the file) or overwrite it (start over at the beginning of the file).

Ctrl-R:Resize Ctrl-V:Move – (lower-middle border) This allows you to move the Browse Window anywhere on the screen or to change the size of the Browse Window to almost any size you want.



Compilers

This section describes how to get the most out of the capabilities in the *iceMASTER* host software in combination with the particular compiler product you are using. *iceMASTER* currently provides full data type support (structures, unions, arrays, etc.) for the compilers listed in this section. More comprehensive support for other compilers will be added in the near future.

Keil/Franklin C51

This information applies to version V3.07 of C51 (C compiler) and version V2.7 of L51 (linker). It should be accurate for later versions of these products and may even be valid for earlier versions.

Recommended Compilation Options

DEBUG	Include debug information in the object file.
OBJECTEXTEND	Include full data type definition debug information in the object file.
NOREGPARMs	Do not pass function parameters in registers.
OPTIMIZE(3)	Do not put auto variables in registers.

Recommended Linking Options

DEBUGLINES	Include source line number debug information in the output file.
DEBUGPUBLICS	Include public (global) symbol information in the output file.
DEBUGSYMBOLS	Include all symbol information in the output file.

Pointers

The Keil/Franklin C51 compiler implements both generic pointers and memory-specific pointers (see page 3-13).

NULL

The value of a NULL generic pointer is a pointer value with all three bytes containing zero (even the selector byte). This is true regardless of whether the pointer is being read (e.g., for comparison to NULL) or written (i.e., NULL being assigned to the pointer).

'typedef' Names

The *iceMASTER* host software is internally set up to display **typedef** names, where appropriate. However, the compiler does not output any information about **typedefs**. The compiler always outputs the underlying type. For example, in

```
typedef struct abc {...} ABC;
ABC xyz;
```

There is no information about 'ABC' in the AOM file (Absolute Object Module file, load module file). The data type of 'xyz' is recorded as '**struct abc**', not '**typedef ABC**'. Likewise, for

```
typedef unsigned char logical;
logical bool_val;
```

there is no information about 'logical' and the type of 'bool_val' is recorded as **unsigned char**.

Enumeration Types

The *iceMASTER* host software is internally set up to display **enumeration** values as they were declared in the C source program (e.g., 'RED', 'GREEN', 'YELLOW', rather than 0, 1, 2). However, the compiler does not output **enumeration** type information. The AOMF-specified type of **enum** variables always seems to be **signed int**.

'double' Types

The data type **double** (double-precision floating-point) is implemented by the compiler as **float** (single-precision floating-point). The data type information in the AOM file for variables declared as 'double' reflects this.

Batch "Make" Files

The batch "make" files TAKF51.BAT and TBKF51.BAT show how the example programs TA.C and TB.C were compiled and linked.

This information applies to version V4.23E/DXT of C-51 (C compiler) and version V4.44D/DXT of XLINK (linker). It should be accurate for later versions of these products and may even be valid for earlier versions.

Recommended Compilation Options

-r0n Put only symbolic information (no source code) into the object file.
Do not add extra NOPs at source line boundaries.

Do not specify '-r1...' or '-r2...': The *iceMASTER* emulators break emulation before the instruction at a break-point is executed. Thus, no extra NOPs are needed in the generated code.

Warning: Do not specify '-r0i...' (add #include file source code to object file) and do not forget to specify the 'n' in '-r0n'. The host software ignores the source images embedded in the load module file. *iceMASTER* locates your original source file on disk and reads that file when displaying source images. Additionally, the line numbers recorded in the resultant load module file (for '-r0i...') will not match the original source file found on your disk.

Recommended Linking Options

-Fdebug Create the load module file in the IAR/Archimedes DEBUG format (.DBG).

Pointers

The IAR/Archimedes C-51 compiler implements only generic pointers; there are no memory-specific pointers (see page 3-13).

NULL

The value of a NULL generic pointer is dependent on the context.

Reading

In comparing a pointer value to NULL, only the offset portion (*hhhh*) is examined (for both bytes being zero). The value in the memory space selector byte is ignored.

Writing

When NULL (zero) is assigned to a pointer, all three bytes in the pointer (including the memory space selector byte) are set to zero.

'typedef' Names

The *iceMASTER* host software is internally set up to display **typedef** names, where appropriate. However, the compiler does not output any information about **typedefs**. The compiler always outputs the underlying type. For example, in

```
typedef struct abc {...} ABC;
ABC xyz;
```

There is no information about 'ABC' in the .DBG file (load module file). The data type of 'xyz' is recorded as **struct abc**, not **typedef ABC**. Likewise, for

```
typedef unsigned char logical;
logical bool_val;
```

there is no information about 'logical' and the type of 'bool_val' is recorded as **unsigned char**.

Enumeration Types

The *iceMASTER* host software is internally set up to display **enumeration** values as they were declared in the C source program (e.g., 'RED', 'GREEN', 'YELLOW', rather than 0, 1, 2). However, the compiler does not output **enumeration** type information. The .DBG-specified type of **enum** variables always seems to be **signed int**.

'double' Types

The data type **double** (double-precision floating-point) is implemented by the compiler as **float** (single-precision floating-point). The data type information in the .DBG file for variables declared as 'double' reflects this.

Batch "Make" Files

The batch "make" files TAIA51.BAT and TBIA51.BAT show how the example programs TA.C and TB.C were compiled and linked.

Appendix A: Source Window & Assembler/Disassembler

Source Window

The **Source Window** is used to display the source program and/or disassembled instructions in Code Memory. The current module (see the *Source/Symbols|Current Module* command) is displayed on the left side of the top border and the filename of the currently loaded program (see the *File|Load* command) is displayed on the right side of the top border.

The information in this window may be displayed in 'Code', 'Mixed' or 'HLL' mode. 'Code' means just an assembly language disassembly of the code will be displayed. 'Mixed' means that HLL source lines (e.g., C language), if available, will be displayed along with the disassembled code. 'HLL' means that just the HLL source lines will be displayed. The default is 'HLL' but can be changed under the *Options* column of the *Configure|Windows|Modify* command.

The starting address of the code displayed is at the PC address and cannot be changed. Note that during single step (*Run|Step*) or slow motion operation (*Run|Slow Motion*) the starting address of the screen does not change until the flow of execution moves out of the code that appears in the window, at which time a new window full of code will be displayed. This is intended to keep the Dynamically Annotated Code visible as long as possible.

In Change Mode, the Source Window also allows you to control the emulation environment.

You can enter Change Mode to browse/peruse the Source Window by pressing the **Tab** key (which is simply a convenient shortcut for the *Configure|Windows|Goto* command) one or more times until the Source Window title is highlighted.

Once you are in Change Mode, by pressing **←**, **→** or **Enter**, the *Source Window Control* Menu becomes available.

Source Window Control				
Browse	Change_Module	Mode	Label_Synch	Set_Break Assemble
Mode=HLL				Label-Synch=On

Source Window Control

The *Source Window Control* Menu is a menu bar window which allows you to browse through or change your code, change the display mode, turn disassembly label synchronization ON and OFF, set or clear simple break-points and set temporary break-points. The commands are:

Browse

The *Browse* command allows you to quickly reposition the highlight bar in the Source Window. If the Source Window display mode is *Code* or *Mixed*, you will be prompted for an address to go to. If the Source Window display mode is *HLL* (source only), you will be prompted for a line number. However, you may enter any address expression (numerically or symbolically) or line number at either prompt. Note that if the display mode is *HLL* and you want to switch to a different module, use the *Change Module* command.

Change Module

In *HLL* (source-only) or *Mixed* display mode, you can use the *Change Module* command to display (switch to) a different module in the Source Window.

Mode

The *Mode* command calls a Pull-down Menu from which you may set the display mode for the Source Window. The available choices are *Code*, *Mixed* and *HLL* (source only).

Mode | *Code*

The *Code* command changes the display mode of the Source Window to a disassembled view of code memory.

Mode | *Mixed*

The *Mixed* command changes the display mode of the Source Window to a disassembled view of code memory, interspersed with HLL source line images for code-generating source lines.

Mode | *HLL*

The *HLL* command changes the display mode of the Source Window to a source-only view of the HLL source module corresponding to the currently highlighted line. This display mode shows both code-generating and non-code-generating source lines.

Label-Synch

The *Label-Synch* command is used to toggle disassembly label synchronization ON and OFF in the 'Code' and 'Mixed' display modes. When Label-Synch is ON, the Host Software uses existing code labels to verify that disassembly addresses are on instruction boundaries. If they are not, addresses will be adjusted so that the disassembly does begin on an instruction boundary. If Label-Synch is OFF, no such checks are made and the disassembly will start at the specified address.

Set Break

The *Set Break* command calls a Pull-down Menu from which you may set or clear simple break-points and set a temporary break-point.

Set Break | *Toggle*

The *Toggle* command toggles (on or off) a permanent simple break-point at the currently highlighted line. Permanent simple break-point locations are flagged with the ▷ character and may be modified in the *Break/Trace* | *Set* menu.

Note that **Ctrl-B** (toggle break-point) performs the same function as the *Set Break* | *Toggle* command.

Set Break | Run Until

The *Run Until* command sets a temporary break-point at the currently highlighted line and then begins emulation immediately from the current PC. This command is essentially the same as the primary *Run | Until* command, except that the temporary break-point address is implicitly denoted as the currently highlighted line in the Source Window.

Note that this "set temporary break-point and start emulation" function may also be accomplished directly from the source window by pressing **Ctrl-T** (set temporary break-point and go).

Assemble

The *Assemble* command invokes the *Assembler/Disassembler* menu (*Display/Alter | Asm/Dasm* command) to allow you to browse through or change (patch) code.

Display/Alter | Asm/Dasm

The *Display/Alter | Asm/Dasm* command calls Assembler/Disassembler Menu Bar Window from which you may assemble instructions into code memory and browse through the disassembled code memory. Browse mode is active on entrance to the *Asm/Dasm* Menu.

```
Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Assembler/Disassembler
Browse Assemble Mode Label-synch Write File:f_demo.aom
Module:F_HLMAIN
Addr Code Label Instruction
00AC 1200EC LCALL WASTETIME ;F_HLMAIN:#38
F_HLMAIN:#39 WasteTime(); /* blank pulse between sets of 5 p
00AF 1200EC LCALL WASTETIME ;F_HLMAIN:#39
F_HLMAIN:#40 } /* end of: for 'counter' */
00B2 0520 INC COUNTER ;F_HLMAIN:#40
00B4 80EF SJMP 00A5h
F_HLMAIN:#42 } /* end of function: 'main()' */
00B6 22 RET ;F_HLMAIN:#42
F_INNER:#14 void InnerLoop( char repeat_cnt )
00B7>8F24 INNERLOOP: MOV REPEAT_CNT,R7 ;F_INNER:#14
F_INNER:#18 for ( i = 0; i < repeat_cnt; i++ ) {
00B9 E4 CLR A ;F_INNER:#18
00BA F522 MOV I,A
00BC C3 CLR C
00BD E524 MOV A,REPEAT_CNT
Mode=Mixed Ctrl-R:Resize Ctrl-U:Move Label-Synch=On
Toggle display mode between Code (instrs only) and Mixed (instrs + HLL images)
sLoad SetBkpt ViewTrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetTg4Go 5GoFrom 6GoUntil7StepIns3StepLin9StepOvr2StepTo
```

Note that the right side of the top border line (under the Menu Bar) shows the filename of any loaded code file, the left side of the bottom border line shows the Display Mode, the center of the bottom border line shows current key information and the right side of the bottom border line shows the Label-synch mode. The commands are:

Display/Alter | Asm/Dasm | Browse

The *Browse* command calls a dialog box from which you may enter an address expression. If an address is entered the display will be repainted starting at the specified address.

Note that if an absolute address is entered and the Label-synch Mode is OFF, the address may or may not correspond to an instruction boundary and may result in an 'out-of-synch' disassembly.

Note for iceMASTER-68HC11 and iceMASTER-68HC05 disassemblies: Several opcode mnemonics have aliases (alternative mnemonic names). In such cases, the alternative name will be shown as a comment next to the disassembled instruction:

68HC11		68HC05	
Mnemonic	Alias	Mnemonic	Alias
ASL	LSL	BCC	BHS
ASLA	LSLA	BCS	BLO
ASLB	LSLB	DECX	DEX
ASLD	LSLD	INCX	INX
BCC	BHS	LSL	ASL
BCS	BLO	LSLA	ASLA
		LSLX	ASLX

You can use either form shown above when patching in new instructions using the single-line assembler (*Display/Alter|Asm/Dasm|Assemble* command).

In addition, for the iceMASTER-68HC11 and iceMASTER-68HC05, disassembled code memory at the current Interrupt Vector area will be displayed as follows:

ivect <address>

The current 68HC11 Interrupt Vector area is determined by the current operating mode of the processor in the probe card. The Special Modes Interrupt Vectors are located at \$BFC0 through \$BFFF. The Normal Modes Interrupt Vectors are located at \$FFC0 through \$FFFF.

The location of the 68HC05 Interrupt Vector area is dependent on the particular chip derivative: \$1FF0-\$1FFF, \$1FF4-\$1FFF, \$1FF8-\$1FFF or \$3FF0-\$3FFF.

Note that interrupt vectors may be set through the *Assemble* command (described below) using

DB \$hhhh

where \$hhhh is the address to put in the vector.

Display/Alter | Asm/Dasm | Assemble

The *Assemble* command turns on Assemble Mode. In this mode you may enter new instructions (and labels) into code memory a single line at a time, at the current (highlighted) address. The address at which to add the instruction may be changed by using *Browse* mode (described above) or by using the ↓ or ↑ keys to increment or decrement the address.

```
Configure File Run Display/Alter Misc Source/Symbols Break/Trace Help
Assembler/Disassembler
Browse Assemble Mode Label-synch Write
Module:F_H New Instruction at 00B7 File:f_demo.aom
Addr Code mov repeat_cnt,5
00AC 1200EC LCALL WASTETIME ;F_HLMAIN:#38
F_HLMAIN:#39 WasteTime(); /* blank pulse between sets of 5 p
00AF 1200EC LCALL WASTETIME ;F_HLMAIN:#39
F_HLMAIN:#40 } /* end of: for 'counter' */
00B2 0520 INC COUNTER ;F_HLMAIN:#40
00B4 00EF SJMP 00A5h
F_HLMAIN:#42 } /* end of function: 'main()' */
00B6 22 RET ;F_HLMAIN:#42
F_INNER:#14 void InnerLoop( char repeat_cnt )
00B7 8F24 INNERLOOP: MOV REPEAT_CNT,R7 ;F_INNER:#14
F_INNER:#18 for ( i = 0; i < repeat_cnt; i++ ) {
00B9 E4 CLR A ;F_INNER:#18
00BA F522 MOV I,A
00BC C3 CLR C
00BD E524 MOV A,REPEAT_CNT
Mode=Mixed Ctrl-R:Resize Ctrl-U:Move Label-Synch=On
Assemble instructions into code memory (patch code memory with new instructions)
sLoad SetBkpt ViewIrc DisAssm MacExec CurMod SrcPath RawSrc SymGlob1SymAlph
1Help 2ResetEm3ResetIg4Go 5GoFrom 6GoUntil7StepIns8StepLin9StepOvr3StepTo
```

When entering an instruction, symbolic information may be used. This includes the ability to define a new label at the current address and to use symbolic names in the operand part of the instruction. When a new label is entered, it is inserted into the internal symbol table as a global (public) symbol.

Note that if the assembled instruction does not contain the same number of bytes as the original instruction at the current address you will be notified and asked to verify the operation through a Confirmation Box. Note that the *Assemble* command is enabled only for a line of machine code, *NOT* for HLL source statements.

To exit Assemble Mode press **Esc** to return to the Menu Bar.

Display/Alter | Asm/Dasm | Mode

The *Mode* command is used to toggle the display mode between 'Code' Mode and 'Mixed' Mode. The 'Code' display Mode means that just assembly language instructions are displayed. The 'Mixed' display Mode (available only if HLL source is available) means that HLL source images are interspersed with the assembly code.

Display/Alter | Asm/Dasm | Label-synch

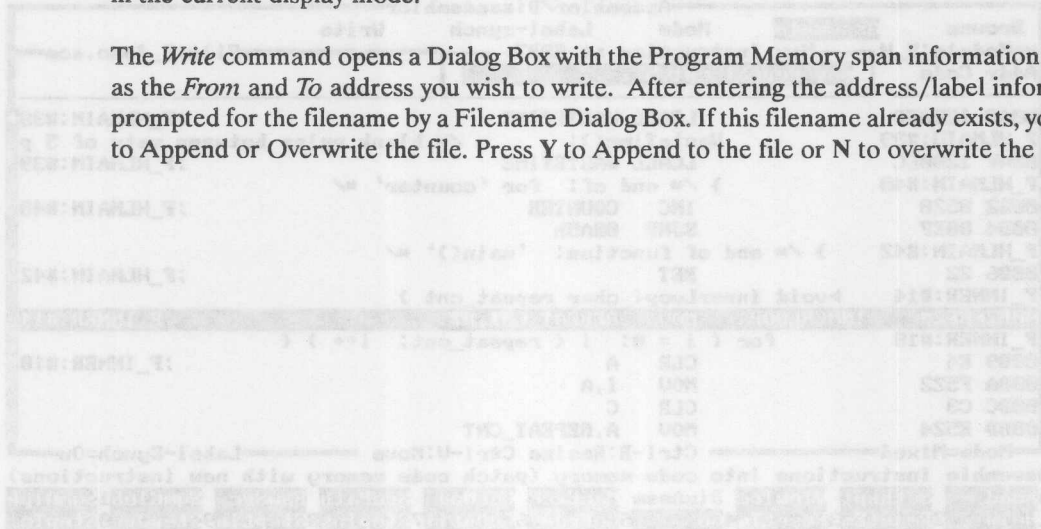
The *Label-synch* command is used to toggle disassembly label synchronization ON and OFF. When Label-synch is ON, the Host Software uses existing code labels to verify that disassembly addresses are on instruction boundaries. If they are not the Host Software will adjust the address so the disassembly begins on an instruction boundary. If the Label-synch mode is OFF, no such checks are made and the disassembly will start at the specified address.

Note that if Label-synch Mode is OFF, the specified address may or may not correspond to an instruction boundary and may result in an 'out-of-synch' disassembly.

Display/Alter | Asm/Dasm | Write

The **Write** command is used to write the disassembled contents of code memory to a disk file. The format of the information written to that file will be just as you see it on the screen (i.e., in human-readable form), in the current display mode.

The **Write** command opens a Dialog Box with the Program Memory span information given to you as well as the *From* and *To* address you wish to write. After entering the address/label information, you will be prompted for the filename by a Filename Dialog Box. If this filename already exists, you will be requested to Append or Overwrite the file. Press **Y** to Append to the file or **N** to overwrite the file.



When entering an instruction, symbolic labels may be used. This includes the ability to define a new label at the current address and to use symbolic names in the operand part of the instruction. When a new label is entered, it is inserted into the internal symbol table as a global (public) symbol.

Note that if the assembled instruction does not contain the same number of bytes as the original instruction at the current address you will be notified and asked to verify the operation through a Confirmation Box. Note that the Assembly command is enabled only for a line of machine code, NOT for HLL source statements.

To exit Assembly Mode press Esc to return to the Main Bar.

Display/Alter | Asm/Dasm | Mode

The **Mode** command is used to toggle the display mode between Code, Mode and Mixed. The 'Code' display mode means that just assembly language instructions are displayed. The 'Mixed' display mode (available only if HLL source is available) means that HLL source images are interspersed with the assembly code.

Display/Alter | Asm/Dasm | Label-synch

The Label-synch command is used to toggle assembly label synchronization ON and OFF. When Label-synch is ON, the Host Software uses existing labels to verify that disassembly addresses are on instruction boundaries. If they are not the Host Software will adjust the address so the disassembly begins on an instruction boundary. If the Label-synch mode is OFF, no such checks are made and the disassembly will start at the specified address.

Appendix B: Virtual Memory Support

Command Line Options

-vme <pages_per_allocation_request>
-vmf <num_bufs> <buf_size>
-vmfn <file_name>

These options control how virtual heap memory will be used. Enabling virtual heap memory allows the Host Software to use memory located outside the conventional 0-640Kb memory area normally usable by a DOS program. This capability is useful if you have a program with a large number of symbols which cannot all fit into the symbol table in conventional memory.

By default, the virtual heap memory is disabled and the Host Software builds the symbol table in conventional memory when loading a program file. If you ever see the following message at the top of the screen

"Not enough unused memory available to continue ... press any key."

the Host Software has used up all the conventional memory and, generally, cannot continue. If this happens, you should reinvoke the Host Software, specifying one or more of the command line options (-vme, -vmf and -vmfn) to enable virtual heap memory usage. However, note that loading files (building the internal symbol table) and some other commands within the Host Software (e.g., listing all symbols alphabetically) may be much slower when virtual heap memory is enabled.

There are two basic kinds of backing store media (real memory/storage) used to support the virtual heap memory:

- 1) LIM EMS 3.2/4.0 Expanded Memory, and
- 2) a file

The fastest medium is expanded memory and you should use this form if:

- 1) You have an expanded memory card in your computer (any type of PC). To allow access to this expanded memory, you must have the appropriate 'DEVICE=..' statement in your CONFIG.SYS file; see the documentation accompanying your expanded memory board for the details on how to do this, or
- 2) You have a '386 or '486 PC with no expanded memory, but with additional memory (extended memory) above the 1Mb boundary. In such a case, you can use the EMM386.EXE device driver supplied with DOS to simulate expanded memory in extended memory. A typical CONFIG.SYS file for this configuration would contain the following lines:

```
DEVICE=C:\DOS\HIMEM.SYS  
DEVICE=C:\DOS\EMM386.EXE
```

You can add additional parameters to the DEVICE command for EMM386.EXE to control the amount of expanded memory to be created (simulated) in your system.

If your computer does not fit either of the two categories above, but it does have extended memory (memory beyond the 1Mb boundary), you can use a file for virtual heap backing store. However, rather than using a file on your hard disk, create a RAM disk in extended memory and, using the '-vmfn' command line option, designate the backing store file as a file on that RAM disk drive. This is tremendously faster than using an actual hard disk file.

-vme <pages_per_allocation_request>

If you have Expanded Memory (LIM-compatible) in your computer, the Host Software can use it as a backing store medium for virtual heap memory. To/ enable access to Expanded Memory, you must have the appropriate 'DEVICE=...' statement in your 'CONFIG.SYS' file. See the documentation accompanying your Expanded Memory board for the specific details on how to do this.

The '-vme <pages_per_allocation_request>' option specifies the number of pages (16Kb per page) of Expanded Memory to request from the Expanded Memory Manager (EMM) each time the Host Software needs more Expanded Memory.

Whenever a program requests some number of logical pages of expanded memory from the EMM, those pages are internally associated with a 'handle'. The total number of available handles, as well as the total number of available logical pages of expanded memory, are resources which could easily be exhausted. If the total number of EMM handles in your system is small, make sure that the value of '-vme <pages_per_allocation_request>' is comparatively large. This will ensure that the available handles are not exhausted before the available logical pages of expanded memory.

To disable use of Expanded Memory as a backing store medium altogether, specify '-vme 0'.

-vmf <num_bufs> <buf_size>

Specifies the number and size of the real-memory buffers to be used to support file backing store as a medium for virtual memory. If either <num_bufs> or <buf_size> is zero, file backing store will not be used at all (i.e., it will be disabled). The integer specified for <buf_size> is the number of K-bytes to allocate for each real-memory buffer.

You can change the path/name of the default backing store file (\$VHEAP\$) using the '-vmfn' option.

To disable use of the file backing store medium altogether, specify '-vmf 0'.

-vmfn <file_name>

If your system has only Extended Memory, but no Expanded Memory, you can significantly speed up access to the backing store file by creating a RAM disk in Extended Memory and then specifying the backing store file name as a file on that RAM disk (e.g., '-vmf E:\TMPFILE', where E: is the RAM disk drive).

Misc | Virtual Memory Command

The *Misc | Virtual Memory* command displays the following information about the current usage of virtual heap memory:

Expanded Memory Backing Store

EMM Version is the version number of the Expanded Memory Manager (EMM) device driver installed in your system. You can use EMM versions 3.2 and beyond.

Page Frame Segment is the segment address of the 64K area called the page frame. This will be somewhere above the conventional memory area used by DOS (0-640K) and below the 1M boundary of the upper memory area. The actual address depends on your computer's configuration.

Mappable Physical Pages is the number of physical pages in the upper memory area (640K-1M) of your computer into which expanded memory pages can be mapped (or swapped). The number here is the sum of the:

- 1) number of pages in the Page Frame (usually 4),
- 2) number of additional mappable pages above the 640K boundary.

Note that your system may have other mappable pages below the 640K boundary, but the Host Software will not use these pages.

This number represents the amount of expanded memory that the Host Software can reference directly at any particular time. The larger the value, the more likely that the memory area/node being referenced is directly accessible, and the less likely that the Host Software will need to swap pages to access the desired memory area/node.

Total Pages is the total number of expanded memory pages in your computer. Each page contains 16K of memory.

Pages Available is the number of pages currently available for allocation by the EMM to the Host Software. The remainder, if any, of the expanded memory pages in your system are unavailable for use by the Host Software because they are allocated to other programs or device drivers.

Pages Allocated to Host Software is the number of pages of expanded memory currently allocated to the Host Software.

Currently In Use by Host Software is the percentage of expanded memory which the Host Software is currently using from the pool of expanded memory pages allocated to the Host Software. When this percentage reaches 100%, the Host Software will request more expanded memory pages from the EMM; the '-vme' command line option (page B-2) specifies how many pages will be requested.

File Backing Store

Real-Memory Buffers is the number of conventional memory buffers being used to swap blocks in from and out to the backing store file. The '-vmf' command line option (page B-2) specifies how many buffers to use.

Buffer (Block) Size is the size of each conventional memory buffer used to swap blocks in from and out to the backing store file. The '-vmf' command line option (page B-2) specifies this buffer size.

The product of Real-Memory Buffers times Buffer (Block) Size represents the amount of real memory that the Host Software can reference directly at any particular time. The larger the value, the more likely that the memory area/node being referenced is directly accessible, and the less likely that the Host Software will need to swap blocks to/from the backing store file to access the desired memory area/node.

File Blocks Allocated is the number of blocks (of size Buffer (Block) Size each) currently needed by the Host Software. Essentially, this is the current size of the backing store file.

Currently In Use by Host Software is the percentage of file backing store which the Host Software is currently actually using. When this percentage reaches 100%, the Host Software will increase the size of the backing store file to create more virtual memory.
